UNIVERSIDAD CARLOS III DE MADRID
DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Aplicación para la desambiguación de entidades en tiempo real usando la plataforma S4

AUTOR:
JAIME DURÁN RODRÍGUEZ

TUTOR:
NORBERTO FERNÁNDEZ GARCÍA

FEBRERO DE 2014

Agradecimientos

Esta memoria pone el punto y final a mi larga etapa como "proyecto de ingeniero", tras muchos meses de esfuerzo y dedicación. No ha sido tarea fácil simultanear esto con un trabajo de 40 horas a la semana y 3 horas de transporte diarias, pero la recompensa de obtener al fin el título y la satisfacción por el trabajo realizado hacen que al final todo haya merecido la pena.

MIL GRACIAS...

... a mis padres por haberme dado la oportunidad que ellos nunca tuvieron de estudiar una carrera, con el esfuerzo económico que eso conllevaba. La mitad del título es mérito vuestro.

... a Miriam por su apoyo y paciencia durante estos últimos meses; gracias por estar ahí siempre y por ayudarme a cerrar esta etapa de mi vida antes de poder abrir otras.

... a mi hermana por los sacrificios derivados de mi capricho de estudiar una ingeniería superior, y por soportar mi mal humor en época de exámenes.

... a mi abuela por existir.

... a mis tíos, primos y amigos por presionarme para acabar de una vez la carrera, aunque algunas veces no hayan colaborado a ello (afortunadamente).

... a toda la gente con la que he tenido la suerte de coincidir en la Carlos III, con una mención especial a los que se convirtieron en amigos, a mis compañeros de prácticas y a la gente solidaria con letra legible.

... y por supuesto a Norberto, por su infinita paciencia y ayuda. ¡Lo hemos conseguido!

Resumen

La clasificación de la información siempre ha desempeñado un papel crucial de cara al análisis, la búsqueda y la indexación de contenidos. Dada la gran cantidad de datos que se pueden llegar a manejar en la actualidad en diversos ámbitos, existe la necesidad de realizar dicha tarea de forma exhaustiva y automatizada. Además, la generación y distribución de la información en tiempo real, hace que dicha tarea se complique bastante, ya que se exigirá al clasificador un procesamiento rápido, fiable y escalable, a la par que eficaz.

En este proyecto se afronta la problemática concreta de desambiguar menciones a entidades ambiguas, presentes en documentos cuya fuente es un flujo en tiempo real, con todas las dificultades que ello supone. El objetivo es implementar una aplicación que consiga solventar dicha problemática mediante aprendizaje, de la forma más eficiente y sencilla posible.

Para resolver el problema de la desambiguación de entidades propiamente dicha, se sondearon los algoritmos de aprendizaje dedicados a clasificación. Debido a su buena relación eficiencia/coste, a su uso extendido en el campo de los clasificadores de texto y a su idoneidad para aplicaciones de tiempo real por su capacidad para aprender de manera incremental, el elegido fue el algoritmo de Naïve Bayes.

Tras llevar a cabo el análisis de distintas alternativas existentes en el estado del arte para el procesado escalable de grandes flujos de información, se seleccionó la plataforma S4 para desarrollar nuestra aplicación, gracias a sus buenas cualidades y al cumplimiento de nuestras necesidades.

A lo largo de esta memoria, se documentan los pasos seguidos para diseñar e implementar la aplicación final, basada en la plataforma S4 y el algoritmo de Naïve Bayes (ambos serán analizados detalladamente). Se explica también el proceso completo de instalación, configuración y ejecución de la aplicación en un entorno distribuido.

La aplicación implementada fue sometida a diversas pruebas para comprobar su correcto funcionamiento y su rendimiento. Entre las conclusiones extraídas de esta evaluación se pueden destacar los buenos resultados obtenidos por el desambiguador, tanto de eficiencia como de fiabilidad; lo que respalda su posible uso en el mundo real. Naïve Bayes y S4 son dos instrumentos muy adecuados para resolver con garantías los problemas planteados al inicio de esta memoria.

Abstract

Classification always has played a key role in data management, in order to analyse, search and index information. Due to the large amount of data we can get to manage these days, the achievement of that task, in an exhaustive and automatic way, is now considered a real need. In addition, real time generation and distribution of information makes our task more complicated, because it requires a fast, reliable and scalable classifier.

In this project, we face a concrete problem within classification process: disambiguation of mentions to entities, in documents coming from a real time input stream. Our main goal is to build an application to efficiently solve this problem via machine learning.

Several learning algorithms were explored, choosing Naïve Bayes for our application because it's a well-known one in text classification field, and adjusts perfectly to our requirements. After analysing existing general purpose tools aimed to build and deploy applications dealing with huge data streams, open source S4 platform was chosen to base our development on, mainly because of its suitability for learning machines, besides a large list of interesting features.

Through this document, we explain all steps followed to design and develop the final application, based on Naïve Bayes algorithm and S4 platform (both will be analysed in detail). Installation, configuration and execution process in a distributed system is explained in depth too.

The resultant application was tested in order to proof its correct behaviour and performance. Among the conclusions extracted from this evaluation, we could highlight the good results obtained by the disambiguator, in efficiency as well as in reliability; which supports its use in real world. Naïve Bayes and S4 are both very suitable to solve with guarantee problems explained at the beginning of this document.

ÍNDICE

Capítulo 1.
Introducción

En la actualidad el mundo de la informática no podría concebirse sin las redes de comunicaciones. El gran referente en este ámbito es sin duda Internet; la red global que ha alcanzado en 2013 la escalofriante cifra de 2.400 millones de usuarios [1]. Podemos afirmar que nuestra vida ha cambiado radicalmente gracias a ella; sentimos la necesidad de estar continuamente conectados para consumir y compartir información en tiempo real.

El dinamismo que ha aportado Internet a nuestros ordenadores (y a un montón de dispositivos como teléfonos, televisores, gafas o relojes) también ha supuesto un cambio enorme en nuestra interacción con los mismos; de hecho es difícil pensar en el uso de dichos dispositivos sin conexión a Internet. Actualmente guardamos nuestros datos en la nube, muchas aplicaciones no funcionan desconectadas, las llamadas y los mensajes de texto ya no son lo más utilizado en los teléfonos móviles, etc. Estos son solo unos pequeños ejemplos de lo que estamos viviendo ahora mismo, y es complicado predecir cómo será nuestra interacción con la red o los dispositivos electrónicos de aquí a pocos años.

La gigantesca cantidad de datos generados en la red de forma continua, implica que su procesamiento para la extracción de información, análisis y clasificación sea la versión moderna del procesamiento por lotes que se realizaba con datos estáticos; este es el gran cambio que ha traído Internet a la minería de datos tradicional. El procesamiento en tiempo real cada vez requiere más recursos, y las aplicaciones dedicadas a esta tarea tienen que lidiar con la inevitable incertidumbre de los flujos de datos. Hasta hace poco, este era terreno exclusivo para caras y complejas aplicaciones propietarias de uso específico, pero en los últimos años han ido apareciendo alternativas gratuitas y

de código abierto para afrontar este problema de forma genérica, facilitando el desarrollo de aplicaciones de usuario en diversos ámbitos.

La motivación de este proyecto es implementar, mediante aprendizaje automático, una aplicación para desambiguación de entidades (nombres de personas, lugares, organizaciones, etc) presentes en un flujo de documentos consumido en tiempo real. Encontrar una solución a este problema resulta interesante porque la cantidad de información que puede llegar a procesarse en la actualidad no para de crecer, y si el procesamiento se hace manualmente no es escalable. En los medios de comunicación y agencias de noticias se ahorraría trabajo a los editores encargados de desambiguar entidades para la clasificación de los artículos. De manera similar se facilitaría la desambiguación en tareas de análisis de medios o en agregadores de noticias.

La dificultad principal a la hora de afrontar el problema reside en el hecho de utilizar un nuevo escenario dinámico, en lugar de tener como entrada los tradicionales datos estáticos. El cambio en la naturaleza de los datos nos obligará a pensar en una nueva estrategia para abordar la desambiguación, lo que nos llevará no sólo a analizar las herramientas disponibles para procesar grandes *streams* de datos, sino también a examinar las distintas estrategias existentes para clasificar texto y desambiguar entidades en tiempo real (los algoritmos más adecuados no son los mismos según tengamos una entrada estática o una dinámica).

En el capítulo a continuación de esta introducción se hace un recorrido por el estado del arte, describiendo al detalle la problemática a la que nos enfrentamos y los requisitos exigidos a nuestra aplicación. Se justifica la elección del algoritmo de aprendizaje Naïve Bayes, necesario para llevar a cabo la desambiguación teniendo en cuenta la naturaleza de los datos de entrada; aparte de esto se explica el algoritmo al completo. Se analizan también las distintas herramientas disponibles para desarrollar aplicaciones destinadas a procesar grandes flujos de datos en tiempo real; profundizando en las características y fundamentos de S4, que será la elegida para llevar a cabo nuestra tarea.

En el capítulo 3 se detalla el proceso de diseño de la aplicación, teniendo en cuenta los requisitos de la misma, así como las características del *framework* elegido para su desarrollo.

En el capítulo 4 se explica la implementación de la aplicación, detallando las particularidades que supone el utilizar S4 para ello. Con el fin de probar la funcionalidad y el rendimiento necesitamos una fuente que genere documentos en tiempo real, como puede ser un medio de comunicación de gran envergadura. Como no tenemos ninguna fuente accesible, usaremos un corpus estático con datos reales

obtenidos del New York Times, que un adaptador convertirá en un flujo de datos para la aplicación.

En el capítulo 5, se exponen todos los pasos necesarios para configurar y ejecutar la aplicación implementada, así como el entorno donde se desplegará.

En el capítulo 6 se lleva a cabo la validación de la aplicación. Para ello primeramente se adjuntan los casos de prueba realizados para testear la funcionalidad. A continuación se extraen a partir de los datos de la entrada unos cuantos ejemplos de ejecución que nos ayudan a obtener resultados sobre la eficiencia de nuestra aplicación con casos reales. Por último se realizan simulaciones de caídas de red o máquinas para ver cómo afectan los problemas a la plataforma, y cómo se recupera la aplicación de los errores.

En el último capítulo se exponen las conclusiones finales del proyecto, y algunas de las posibles líneas futuras de trabajo.

Para finalizar esta memoria se adjuntan varios anexos donde se aporta más información sobre determinados aspectos que podrían ser interesantes para el lector y que forman parte del trabajo realizado durante la ejecución del proyecto.

Capítulo 2.
Estado del arte

En este capítulo se detalla la problemática que se quiere abordar con la ejecución del presente proyecto, y se analizan las distintas opciones disponibles para llevarlo a cabo. Después de elegir la manera de afrontar el problema, se analiza la solución escogida para poner de manifiesto su potencial, con la vista puesta en las fases de diseño e implementación.

2.1. DESCRIPCIÓN DEL PROBLEMA

La clasificación de la información desempeña desde hace tiempo un papel esencial de cara a su posterior análisis, indexación y búsqueda. La gran cantidad de datos que se manejan en la actualidad en numerosos ámbitos obliga a buscar herramientas que automaticen y optimicen en la medida de lo posible dicho proceso de extracción de la información.

En el ámbito de los medios de comunicación y las agencias de noticias existe la necesidad de clasificar la información en base a determinados parámetros; como pueden ser la temática o categoría del artículo, las etiquetas que se pueden asignar al mismo, o las entidades con nombre [2] (también llamadas simplemente entidades) presentes en el mismo. Dichas entidades pueden ser desde lugares a personas; pasando por organizaciones, fechas, monedas, etc.

El proceso de clasificación en este ámbito se lleva a cabo típicamente gracias a un editor, que asigna a cada artículo la categoría o las etiquetas, e identifica las entidades presentes, entre otras cosas. Este tipo de clasificación permite a posteriori por ejemplo

presentar la información organizada temáticamente, ofrecer artículos relacionados con el que estamos leyendo, o analizar y comparar las noticias de distintos medios.

El presente proyecto afronta una tarea concreta dentro de la problemática generalizada de la extracción de información: la desambiguación de entidades con nombre, en un contexto especial donde los documentos provienen de una o varias fuentes de información en tiempo real, con los problemas añadidos que esto supone. El objetivo final es implementar una aplicación eficiente y fiable que consiga ayudar a solventar dicho problema por medio de aprendizaje automático [3], de tal forma que se ahorre tiempo y esfuerzo a los editores.

En la actualidad los medios de comunicación y las agencias de información generan una gran cantidad de contenidos. El problema reside precisamente en que cuando el proceso de desambiguación de entidades se realiza de forma manual no resulta escalable; mayor cantidad de información requerirá mayor esfuerzo por parte de los editores. Además los medios y las agencias cuentan con novedosos canales de distribución, como pueden ser las redes sociales o el *microblogging* [4]; ejemplos de ello son Twitter [27] o Eskup [28]. Esto implica que existan aún más fuentes de información y con mayor dinamismo si cabe.

Como entrada a nuestro sistema contaremos con un flujo de documentos o artículos provenientes de un medio informativo. La información relevante en estos documentos serán sus metadatos ya extraídos: información añadida sobre categorías, entidades presentes, etiquetas, etc.

Nuestro problema a resolver, explicado de forma genérica, aparece cuando recibimos un documento en el que tenemos una mención que no se ha podido asociar a una entidad concreta, ya sea porque no la conocemos o porque dicha mención es ambigua. Por ejemplo, en un artículo puede aparecer la cadena de texto “Alonso” de forma que posiblemente no se pueda identificar automáticamente la entidad a la que corresponde, ya que la mención podría referirse a distintas entidades, como podrían ser:

- Fernando Alonso; piloto de Fórmula 1
- Xabi Alonso; futbolista

Si este mismo artículo incluyera la entidad con nombre “Ferrari”, un lector con conocimientos suficientes sobre el tema podría suponer que la mención anterior pertenece a la entidad con nombre “Fernando Alonso” (piloto de carreras), teniendo una alta probabilidad de acierto. De la misma manera, si el artículo perteneciera a la categoría “fútbol”, el lector podría intuir que la mención podría referirse a la entidad con nombre “Xabi Alonso” (futbolista).

Nuestra aplicación no contará con conocimientos iniciales de ningún tipo, y por eso hay que someterla a un aprendizaje previo, donde se preparará para enfrentarse a reconocimiento de entidades en ambigüedades como la anterior, resolviendo las menciones de la mejor forma posible e intentando mejorar la efectividad a medida que su entrenamiento vaya en aumento. Esta tarea se basará en un algoritmo de aprendizaje adecuado a los requisitos de la aplicación.

Para llevar a cabo el diseño, nos centraremos en varios puntos claves:

- El análisis del contexto del documento; es decir, de todos los metadatos disponibles (la entrada a nuestra aplicación).
- La memoria de nuestro clasificador; o lo que es lo mismo, las estadísticas obtenidas gracias a los metadatos de los documentos procesados con anterioridad sin ambigüedades (aprendizaje o entrenamiento).
- La estimación de probabilidades para las distintas entidades candidatas a partir de las estadísticas anteriores (desambiguación).

El hecho de que la fuente de datos sea un flujo de documentos en tiempo real, a priori de gran envergadura, hace que nuestra tarea se complique bastante, ya que la aplicación resultante deberá cumplir con una serie de exigentes requisitos; entre ellos tener una gran capacidad de procesamiento, eficiencia, bajo retardo, fiabilidad y escalabilidad. Esto marcará por completo el diseño y la implementación de la aplicación.

Para llevar a cabo el proceso de desambiguación existen distintos algoritmos de aprendizaje automático disponibles, que analizaremos en el siguiente apartado con el fin de saber si hay alguno que se ajuste a los requisitos de la aplicación.

Como segundo paso hacia el diseño de la aplicación, analizaremos varias herramientas genéricas para desarrollo y despliegue de aplicaciones que estén destinadas a consumir grandes flujos de datos en tiempo real. Elegiremos la más adecuada conforme a nuestras necesidades.

2.2. ALGORITMO DE APRENDIZAJE

Para llevar a cabo la implementación de nuestra aplicación con aprendizaje automático, es primordial elegir el algoritmo que usaremos a la hora de estimar la probabilidad de que una mención ambigua esté asociada a una entidad u otra.

La necesidad en nuestro caso de utilizar un algoritmo de desambiguación adecuado para datos de entrada en tiempo real, provoca que exista una restricción sobre el uso

de determinados algoritmos que sólo pueden ser utilizados para procesar datos estáticos.

Dentro de los algoritmos que pueden usarse en tiempo real, existe una larga lista, donde destacan algunos como SVM (Máquinas de Vectores Soporte) [5] o Random Forests (basado en Árboles de Decisión) [6]. Existe un requisito que nos impedirá usar estos dos populares algoritmos, y es que necesitamos que la aplicación funcione en modo *streaming*; el algoritmo elegido debe poder aprender de forma dinámica, ya que los documentos de entrenamiento llegarán intercalados con los que requieran desambiguación de una entidad.

Una vez acotado el tipo de algoritmo que se necesita, contamos con varias alternativas; por un lado está el popular algoritmo Naïve Bayes [7], que como su propio nombre indica se trata de un clasificador bayesiano. Este tipo de clasificadores es uno de los métodos más utilizados a la hora de implementar aplicaciones con aprendizaje máquina en tiempo real, especialmente en el campo de los clasificadores de texto. Por otro lado tenemos algoritmos menos conocidos, pero no por ello obtienen peores resultados; por ejemplo los Árboles de Hoeffding [8] o SPDT (*Streaming Parallel Decision Tree*) [9].

El requisito de poder procesar un gran volumen de artículos en tiempo real, hace imprescindible que el algoritmo elegido sea lo menos complejo posible, ya que su tiempo de procesamiento deberá ser extremadamente rápido. Gracias a su gran relación eficiencia/coste, a su máxima sencillez, y a su extendido uso en el ámbito de la clasificación de texto, elegimos para la implementación de nuestra aplicación una versión del algoritmo de Naïve Bayes. Dicha solución se utilizará de cara a estimar la probabilidad de las distintas entidades candidatas a resolver una ambigüedad. Vemos a continuación el desarrollo del algoritmo.

En nuestro desambiguador se define como entrada un conjunto de entidades candidatas:

$$X = \{x_i\}$$

y el conjunto de metadatos de un documento **d**, que se puede notar como:

$$d = \{f_j\}$$

Nuestra máquina tendrá que estimar cuál es la entidad $\mathbf{x_i}$ más probable para resolver la ambigüedad que se presenta, dado el contexto del documento. Podemos representar dicha entidad como un máximo a posteriori:

$$x_{map} = \underset{x_i \in X}{\arg max}\ P(x_i|d)$$

El Teorema de Bayes en probabilidad nos ayuda a desarrollar la fórmula anterior:

$$x_{map} = \underset{x_i \in X}{\arg max} \frac{P(d|x_i) \cdot P(x_i)}{P(d)}$$

Como buscamos el valor de $\mathbf{x_i}$ que maximice la expresión, podemos eliminar el denominador:

$$x_{map} = \underset{x_i \in X}{\arg max} P(d|x_i) \cdot P(x_i)$$

Sustituyendo **d** por su expresión compuesta obtenemos:

$$x_{map} = \underset{x_i \in X}{\arg max} P(f_1, f_2, \dots, f_n|x_i) \cdot P(x_i)$$

Aplicamos ahora Naïve Bayes, haciendo las siguientes suposiciones:

- El orden de las entidades no importa.
- Todas las probabilidades $P(f_j|x_i)$ son independientes entre sí dado el documento **d**. Por esto precisamente se llama algoritmo de Bayes "ingenuo".

Obtenemos finalmente:

$$x_{nb} = \underset{x_i \in X}{\arg max} P(x_i) \cdot \prod_{f_j \in d} P(f_j|x_i)$$

Podremos estimar las probabilidades que aparecen gracias a las estadísticas tomadas durante el aprendizaje de nuestra máquina, solucionando así el problema con simples contadores internos en los PE implicados.

Las probabilidades estimadas quedarían de la siguiente forma:

$$\hat{P}(x_i) = \frac{N(x_i)}{N(d)}$$

$$\hat{P}(f_j|x_i) = \frac{N(f_j, x_i)}{N(x_i)}$$

Siendo:

- $N(x_i)$: número de apariciones de la entidad $\mathbf{x_i}$ dentro de los documentos procesados en modo de entrenamiento.
- $N(d)$: número de documentos procesados en modo de entrenamiento.
- $N(f_j, x_i)$: número de apariciones conjuntas de la entidad $\mathbf{x_i}$ y la entidad $\mathbf{f_j}$ en modo de entrenamiento

Para evitar el caso en que no tengamos ninguna aparición previa en entrenamiento de $\mathbf{x_i}$ o de $\mathbf{f_j}$, (provocando que el productorio fuese nulo), así como el caso en que el número de documentos procesados sea 0 (creando divisiones entre 0), podemos aplicar una generalización del suavizado de Laplace:

$$\hat{P}(x) \approx \frac{N(x) + \alpha}{N(d) + \alpha n}$$

$$\hat{P}(f|x) \approx \frac{N(f,x) + \alpha}{N(x) + \alpha n}$$

Siendo **n** el total de entidades distintas que han aparecido en todos los documentos procesados en entrenamiento, y $\boldsymbol{\alpha}$ un factor constante arbitrario para realizar el ajuste. Cabe señalar que en el caso de $\hat{P}(x)$ se ha aplicado el mismo suavizado que para $\hat{P}(f|x)$; aunque existen otras variantes, esta fue la elegida empíricamente.

La constante $\boldsymbol{\alpha}$ vale 1 en el suavizado de Laplace original; valor que acabamos adoptando para nuestra aplicación después de haber probado otros con peores resultados.

La expresión final resulta:

$$x_{nb} \approx \underset{x_i \in X}{\arg max} \frac{N(x_i) + 1}{N(d) + n} \cdot \prod_{f_j \in d} \frac{N(f_j, x_i) + 1}{N(x_i) + n}$$

Existe el peligro de que el productorio que aparece en la fórmula resultado de aplicar Naïve Bayes devuelva valores muy pequeños, llegando incluso a ser 0 por problemas de resolución numérica. Una solución para evitar este problema es usar la versión logarítmica del algoritmo:

$$x'_{nb} = \underset{x_i \in X}{\arg max} \left(\ln\big(P(x_i)\big) + \sum_{f_j \in d} \ln\Big(P\big(f_j|x_i\big)\Big) \right)$$

Como lo importante es la entidad que nos dé el valor máximo y no el rango donde se mueva el resultado, esta aproximación es igual de válida que la anterior.

La expresión final para esta versión del algoritmo queda así:

$$x'_{nb} \approx \underset{x_i \in X}{\arg max} \left(\ln\left(\frac{N(x_i) + 1}{N(d) + n}\right) + \sum_{f_j \in d} \ln\left(\frac{N(f_j, x_i) + 1}{N(x_i) + n}\right) \right)$$

Para asegurarnos de que no existe ningún problema con cualesquiera que sean los datos de entrada, es recomendable usar la versión logarítmica del estimador, que se

mueve en un margen de números mucho mayores. Esta versión penaliza el rendimiento de la aplicación por el uso de logaritmos en una cantidad de tiempo insignificante.

2.3. PROCESADO DE DATOS EN TIEMPO REAL

Los exigentes requisitos ligados a toda aplicación que trabaja con una gran cantidad de datos en tiempo real, hacen necesario que todas las etapas de su ciclo de vida, desde el diseño hasta su puesta en funcionamiento, estén orientadas hacia el cumplimiento de ciertos mínimos; como pueden ser la provisión de un mecanismo de recuperación ante errores, una alta eficiencia, bajo retardo, o la escalabilidad del entorno donde se instala.

La principal dificultad a la hora de procesar flujos de datos en tiempo real reside en que no se conoce a priori la tasa de llegada de los mismos; ni se puede predecir, ni se puede controlar. La aplicación tendrá que lidiar con los incrementos (y decrementos) de la tasa de datos de entrada, teniendo incluso que llegar a eliminar eventos mediante la técnica conocida como desprendimiento de carga [10].

Para resolver el problema de procesar grandes cantidades de datos de forma distribuida, surgieron hace ya varios años distintas plataformas dedicadas básicamente a ello. En la actualidad la más extendida es Apache Hadoop [29]; una herramienta de código abierto que da soporte completo a aplicaciones destinadas a trabajar con miles de nodos y *petabytes* de datos [11].

Hadoop se basa en el modelo de programación MapReduce [12]. Dicho modelo permite, de manera sencilla, la implementación de tareas de procesamiento de datos por lotes de forma distribuida. El funcionamiento resumido consiste en que los datos de entrada se dividen en pares clave-valor que son procesados por una función *map*, produciendo nuevos valores intermedios que son a continuación agrupados por clave, y procesados en conjunto mediante otra función *reduce*. Aparte de esto, MapReduce permite la posibilidad de operar en grandes clústeres (conjuntos de máquinas) sin preocuparse por problemas del sistema. Su diseño se pensó para resolver problemas en el contexto de aplicaciones de búsqueda basadas en minería de datos (extracción de información “oculta” o patrones, que residen de manera implícita en los datos) y algoritmos de aprendizaje máquina.

Con la aparición de proyectos de código abierto como Hadoop, la adopción del modelo MapReduce se extendió, y pasó a usarse en escenarios reales; tales como búsquedas web, detección de fraudes, y webs de citas online.

Aunque hace ya tiempo desde que el *software* orientado al procesamiento de datos por lotes alcanzase su madurez, hasta principios de esta década no existían tales avances en el ámbito de los grandes flujos de datos procesados en tiempo real.

La aparición en su día de ciertas aplicaciones, como fueron las búsquedas en tiempo real en Internet, la negociación de alta frecuencia en los mercados financieros, o las redes sociales, puso de manifiesto la necesidad de crear soluciones altamente escalables para procesar grandes flujos de datos. La investigación en este campo comenzó a finales de la década de los 90, y con los años fueron surgiendo algunos proyectos y motores comerciales, pero su uso estaba restringido siempre a un campo muy concreto. Al tiempo comenzó a gestarse la creación de plataformas *middleware* de uso genérico, que permitieran a los desarrolladores centrarse en la funcionalidad de sus aplicaciones. La intención era conseguir algo como Hadoop para flujos de datos en tiempo real. De hecho, una primera aproximación a dicho propósito fue concebida modificando el propio Hadoop, e incluso más adelante se llegó a trabajar en un prototipo llamado HOP (Hadoop Online Prototype) [13]. Esta solución, al igual que otras similares, implementaba la estrategia de segmentar los flujos de datos en trozos de longitud fija que son procesados a posteriori por el algoritmo MapReduce. La principal desventaja de este procedimiento es que para elegir el tamaño de los segmentos hay que recurrir a una solución de compromiso; si se intenta reducir su tamaño, se obtendrán más segmentos, lo que supondrá tener que introducir más información de cabeceras, haciendo el procesamiento más costoso. Si por el contrario se aumenta el tamaño de los segmentos, aumentará el retardo en el sistema.

Algunas de las primeras plataformas middleware en aparecer fueron STREAM [30], Borealis [31], Aurora [32] o Telegraph [33]. Todas estas limitaban bastante el tipo de aplicaciones que podían soportar, pero con el tiempo fueron apareciendo otras más sofisticadas.

En 2006 la compañía IBM presenta SPC (Stream Processing Core) [14], una plataforma distribuida, diseñada para soportar aplicaciones destinadas a procesar información procedente de grandes flujos de datos, y que resolvía algunas de las carencias de sus predecesores. El modelo de programación de SPC fue el primero en soportar operaciones definidas por el usuario, yendo más allá de los operadores relacionales. Otra de sus novedades con respecto a otros sistemas ya existentes fue la forma de definir los flujos de datos, usando una especie de modelo de suscripción donde cada elemento de procesado determina las propiedades deseadas para filtrar los flujos de datos que le interesan. Además de todo lo anterior, el diseño de la arquitectura de SPC puso especial énfasis en la eficiencia y la escalabilidad.

En 2008 la división Labs de Yahoo llevó a cabo un proyecto de investigación en el ámbito de la publicidad en internet. Los motores de búsqueda comerciales, como

Google o Yahoo, devuelven desde hace años anuncios textuales junto a los resultados de las búsquedas, obteniendo unas ganancias directamente proporcionales al número de clics que se realizan sobre dichos anuncios. Con el fin de mostrar los anuncios más adecuados e interesantes para cada usuario, y en una posición óptima dentro de la página, Yahoo desarrolló unos algoritmos que estiman dinámicamente la probabilidad de un clic sobre un determinado anuncio y en un determinado contexto. Este contexto incluye preferencias de usuario, ubicación geográfica, búsquedas, clics anteriores, etc; lo que constituye una gran cantidad de información teniendo en cuenta el número de usuarios del buscador. Por ello Yahoo puso énfasis en el diseño y arquitectura de la propia plataforma, de manera que se adaptase a grandes flujos de datos con bajo retardo y de forma escalable. Esta aplicación fue el embrión de lo que en 2009 se dio a conocer como S4 (Simple Scalable Streaming System) [15][16]; una plataforma de código abierto que permite desarrollar aplicaciones para el procesamiento de flujos ilimitados de datos, de una forma distribuida y escalable.

La aparición de S4 obedece a la necesidad de rellenar el hueco existente entre los complejos sistemas propietarios de funcionalidad similar, y las plataformas de código abierto especializadas en el procesado en lotes de grandes cantidades de datos estáticos; donde tenemos Apache Hadoop como principal referencia.

S4 comparte en sus inicios muchos puntos en común con SPC. Ambas plataformas fueron diseñadas para procesar grandes flujos de datos, y son capaces de extraer información usando operaciones definidas por los programadores de las aplicaciones. Las principales diferencias residen en su arquitectura; mientras el diseño de SPC se basa en un modelo de suscripción, el de S4 proviene de una combinación de MapReduce con el Modelo de Actores [17]; en respuesta a un mensaje recibido, un elemento (actor) del sistema puede tomar distintas decisiones de forma asíncrona: realizar acciones locales, crear más actores, enviar mensajes, o determinar la forma de responder al siguiente mensaje recibido.

El nivel de simplicidad alcanzado por el diseño de S4 es mayor que el de SPC debido a su simetría: todos los nodos en el clúster son idénticos y no existe un control centralizado gracias al uso de Apache Zookeeper [34] un servicio de gestión de clústeres; lo que hace que su eficiencia y su escalabilidad sean mayores. S4 además aspira a esconder al programador la complejidad asociada al procesamiento en paralelo, haciendo el desarrollo y el despliegue más simples.

A finales del año 2011, Twitter liberó el código de una plataforma dedicada al procesado de eventos en tiempo real, y desarrollada por la recién adquirida Backtype [35]. Dicha plataforma, denominada Storm [36], había sido diseñada precisamente para procesar la información proveniente de miles de millones de *tweets*. El hecho de

tener que lidiar con semejante cantidad de datos, hizo que su diseño se enfocase en perseguir las mismas metas que S4.

La filosofía de Storm es distinta a la seguida por SPC y S4, aunque sus resultados son similares. El modelo de programación de Storm no sigue un paradigma concreto como los anteriores; una aplicación en Storm define una topología, mezcla de *spouts* (fuentes de flujos de datos) y *bolts* (elementos que procesan y emiten flujos). Los *bolts* procesan los eventos que recogen de sus fuentes, que pueden ser tanto *spouts* como *bolts*. Cabe destacar que con esta topología podría implementarse fácilmente la funcionalidad de MapReduce, usando un *bolt* para la función *map* y otro para la función *reduce*.

Desde su lanzamiento, *Storm* se convirtió en una herramienta puntera en este campo, aunque todavía, dos años después, no podamos afirmar que sea "el Hadoop del procesado en tiempo real" como esgrimían sus creadores. De hecho a día de hoy todavía siguen apareciendo nuevas plataformas de este tipo, como por ejemplo Apache Samza [37], creada y usada por Linkedin [38], y liberada en septiembre de 2013. Esta plataforma no alcanza el nivel de madurez de S4 y de Storm, pero comparte la mayoría de sus características y ha sido probada al igual que ellas en un entorno exigente. La aparición de novedades en este campo sugiere que todavía existe margen de mejora, aunque el hueco que existía ya se pueda dar por cubierto.

Para elegir la plataforma de desarrollo, en el momento de arranque de este proyecto tanto Storm como S4 cumplían con los requisitos que se pueden exigir a nuestra aplicación para reconocimiento de entidades con nombre, y partían con ventaja porque eran las dos más maduras y extendidas.

La elegida después analizar pros y contras, así como las diferencias entre ambas, fue S4, por los siguientes motivos:

- Es a priori la que mejor se ajusta a lo que necesita una aplicación de procesamiento de texto, debido a su modelo de programación mezcla entre MapReduce y el Modelo de actores.
- Otorga a las aplicaciones la posibilidad de recuperar el estado de un nodo en caso de errores, mientras que Storm simplemente garantiza el procesamiento de todos los datos. Ambas son perfectamente válidas para prevenir errores, pero la opción de S4 es la ideal para aplicaciones de aprendizaje máquina como la nuestra, ya que es esencial no perder lo ya aprendido.
- Da facilidades al programador en las fases de implementación y despliegue, aportando un *framework* completo en lugar de las herramientas para crear uno (Storm) [18].

A favor de Storm el mejor argumento que teníamos es que su comunidad era mayor y más activa, por lo que el soporte y la evolución podían ser mejores (aunque en la experiencia con S4 no hemos detectado carencias en ese apartado).

Queda fuera del alcance de este proyecto la comparación exhaustiva entre ambas plataformas, sobre todo porque ambas se encuentran en constante evolución y han ido mejorando desde nuestra elección. De todas formas, recomendamos la lectura de dos comparativas bastante completas realizadas durante la ejecución del presente proyecto; una de Gianmarco de Francisci (colaborador en S4 y usuario de Storm) en su blog [18], y otra de Richard McCreadie para la Universidad de Glasgow [19].

A continuación se profundiza en los fundamentos del *middleware* elegido.

2.4. ANÁLISIS DE S4

En este apartado se hace un breve análisis de la plataforma S4; algo importante de cara a la implementación de nuestra aplicación, y también interesante para entender parte de su funcionamiento. Es aquí donde se explican las características y peculiaridades genéricas de S4 que luego se verán reflejadas en el desambiguador.

2.4.1. Características principales

La plataforma elegida como base para nuestra aplicación cuenta con las siguientes señas de identidad y atractivos [15][39]:

- **Abierto**: S4 comenzó como un proyecto de investigación en Yahoo! Labs en agosto de 2008. Pasó a ser de código abierto en septiembre de 2009, y migró a la Incubadora de Apache en septiembre de 2011, operando bajo la licencia de Open Source Apache 2.0 [40]. Esto significa que su código fuente puede ser modificado y distribuido respetando los términos de la licencia.
- **Escalable**: El rendimiento de S4 se incrementa linealmente añadiendo nodos adicionales al clúster. No se especifica un límite máximo de nodos soportados a priori.
- **Descentralizado**: Todos los nodos del sistema son simétricos; es decir, comparten la misma funcionalidad y responsabilidad sin la existencia de ningún servicio centralizado, lo que elimina los fallos globales y simplifica los despliegues, el mantenimiento y los cambios en el clúster.
- **Accesible para el desarrollador**: Es sencillo implementar y desplegar aplicaciones usando la API de S4. Para facilitar la tarea al desarrollador, se incluyen varias aplicaciones básicas con el propio software.

- **Extensible**: S4 ofrece la posibilidad de realizar implementaciones a medida de los distintos bloques que forman la plataforma.
- **Gestión transparente del clúster**: S4 oculta todas las tareas de gestión del clúster bajo una capa de comunicación que se basa en Apache Zookeeper; un servicio de coordinación para aplicaciones distribuidas también de código abierto.
- **Tolerancia ante fallos:** S4 ofrece un mecanismo de respuesta ante errores para conseguir alta disponibilidad. Cuando surge un problema en un nodo del clúster, otro nodo en espera se activará automáticamente para retomar sus tareas. S4 nos da además la posibilidad de activar copias de seguridad automáticas para ir guardando los datos importantes de los nodos, de tal forma que la pérdida de información sea mínima en caso de haber problemas con uno de ellos.
- **Probado**: S4 ha sido desplegado en el entorno de producción de Yahoo, llegando a procesar miles de peticiones por segundo.
- **Minimización del retardo**: Se usa memoria local en cada nodo, evitando así los cuellos de botella en la escritura/lectura de disco.

Otras características interesantes:

- **Despliegue flexible de aplicaciones:** Los paquetes de las aplicaciones son ficheros JAR estándar con la extensión .s4r, que se crean en una máquina, se suben a una ubicación accesible para el clúster, y automáticamente se despliegan en todos sus nodos. Los propios módulos de la plataforma también son ficheros JAR estándar.
- **Diseño modular:** Tanto la plataforma como las aplicaciones están construidas mediante inyección de dependencias [20], y se configuran a través de módulos independientes. Por ejemplo, es posible mezclar varias políticas de servicio de eventos: desprendimiento de carga, regulación de la frecuencia, o bloqueo. Su diseño es lo más genérico, ampliable y adaptable posible.
- **Acoplamiento dinámico simple de aplicaciones:** Conseguido gracias a un mecanismo de publicación-suscripción [21] donde el encargado de enviar información no conoce al destinatario; y los interesados en determinado tipo de información se suscriben a ella, sin saber quién la genera. Esto permite ensamblar subsistemas en mayores plataformas (con un control independiente sobre cada uno de ellos), reutilizar aplicaciones, o separar el pre-procesamiento de la aplicación en sí.
- **Balanceo de carga automático:** S4 hace balanceo de carga estático entre nodos de forma transparente para el desarrollador.

Algunos de los usos de S4 hasta la fecha han sido por ejemplo [16]:

- Detección de intrusiones en una red.
- Predicción de precios en mercados financieros.
- Personalización de las búsquedas.
- Extracción de temas de moda en Twitter.
- Optimización de parámetros de una plataforma en tiempo real.
- Filtrado de correo basura.
- Monitorización de redes.

2.4.2. Fundamentos

Un sistema S4 se compone básicamente de **Elementos de Procesado** (**PE**). Para una aplicación específica se definirán distintos tipos de estos elementos (con distinta funcionalidad y cometido) que se corresponden con clases Java. A medida que vayan llegando datos a un tipo de PE se irán creando dinámicamente instancias según se vayan necesitando, a partir de un objeto especial conocido como prototipo del PE.

Los PE están destinados a consumir **eventos** de entrada. Dichos eventos contendrán una clave relacionada con los datos que transportan; es lo que se conoce como eventos de datos con clave.

Cada PE del sistema llevará a cabo alguna(s) de las siguientes acciones después de consumir y procesar el evento de entrada:

- Lanzar uno o más eventos que serán consumidos por otros PE.
- Guardar o publicar resultados.

Como vimos anteriormente, el modelo de programación de S4 es una combinación de MapReduce y el modelo de Actores. S4 adopta dicho modelo como referencia ya que se trata de una arquitectura distribuida que intenta evitar el uso de memoria compartida entre los servidores del clúster. Además es un modelo bastante simple y probado en varios *frameworks* a gran escala [22].

En S4 los cálculos y operaciones son realizados por los PE, y los mensajes son transmitidos entre ellos en forma de “eventos con datos”. El estado (datos internos) de un PE es desconocido para el resto; el envío y consumo de eventos es la única forma de interacción entre ellos. S4 proporciona a las aplicaciones la capacidad de enrutar los eventos a los PE apropiados, y la posibilidad de crear nuevas instancias de esos tipos de PE; todo ello de forma transparente para el desarrollador.

Los conceptos principales que se manejan en la plataforma S4 son:

- **PE**: elemento básico de procesado dentro de la aplicación.
- ***Stream***: flujo de datos entre dos tipos distintos de PE o entre dos aplicaciones.
- **Instancia de PE**: objeto de la clase perteneciente a un determinado tipo de PE.
- **Evento**: objeto que se envía por un *stream* y que lleva información para la instancia de PE destino.
- **Aplicación**: es la que engloba y relaciona los tipos de PE (clases), los *streams* que establecen la comunicación entre ellos, y los prototipos de los eventos que se envían por esos *streams* (también clases).
- **Nodo**: lugar donde se crean las instancias de los prototipos de PE de la aplicación desplegada.
- **Clúster**: conjunto de nodos que van a ejecutar una misma aplicación. Ofrece un número determinado de particiones libres, también llamadas tareas, que son ocupadas por nodos a los que se denomina activos; el resto, de haberlos, serán nodos en espera.

A continuación se profundiza en los elementos más importantes de la arquitectura de S4.

2.4.2.1. Elementos de procesado

Los Elementos de Procesado (PE) son las unidades de procesamiento computacional básicas en S4. Cada instancia de un PE es identificada de forma unívoca por 4 propiedades:

1. Funcionalidad (definida por la implementación de una clase o prototipo de PE y su configuración asociada)
2. Tipo de eventos que consume (definido por el flujo al que se suscribe el PE y la clase asociada al evento)
3. Identificador de la clave de esos eventos consumidos
4. Valor de dicha clave

Cada PE consume solamente los eventos de un determinado flujo al que se suscribe, y cuya clave (identificador y valor) se corresponde con la que se le designó inicialmente. Una nueva instancia de PE será creada automáticamente para cada valor nuevo de la clave, partiendo siempre del prototipo de dicho PE. También podrían configurarse de tal forma que se cree sólo una instancia para procesar todos los eventos del *stream* de entrada independientemente de su clave.

El proceso anterior se detalla en las siguientes ilustraciones:

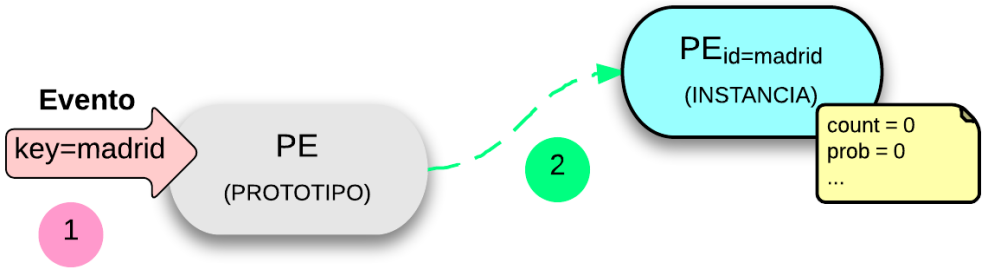

Ilustración 1. Funcionamiento de PEs ante eventos (1/4)

1) Llega el primer evento al prototipo del PE, con clave “madrid”.
2) Se crea una instancia del PE a partir del prototipo, inicializando sus variables internas.

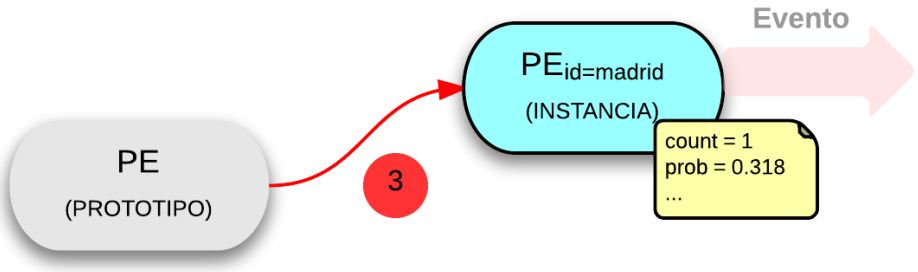

Ilustración 2. Funcionamiento de PEs ante eventos (2/4)

3) El evento pasa a la instancia recién creada, que será la encargada de procesar sus datos y actualizar variables, guardar resultados, o generar nuevos eventos.

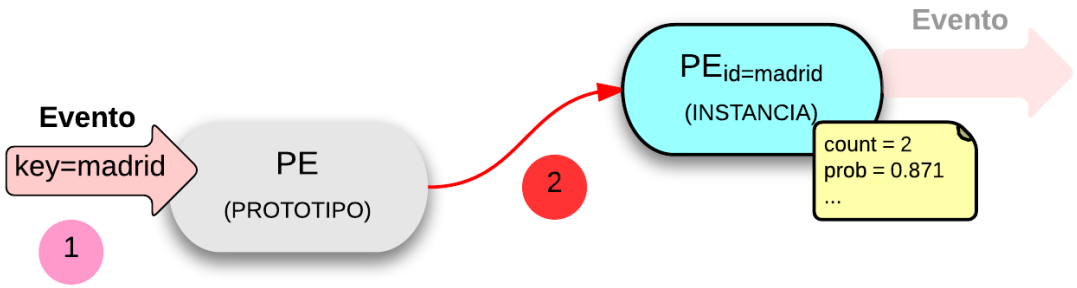

Ilustración 3. Funcionamiento de PEs ante eventos (3/4)

1) Llega un nuevo evento con la misma clave que antes.
2) La aplicación detecta que ya existe una instancia de ese PE para dicha clave, por lo que le manda el evento. Dicha instancia lo procesa igual que hizo con el anterior.

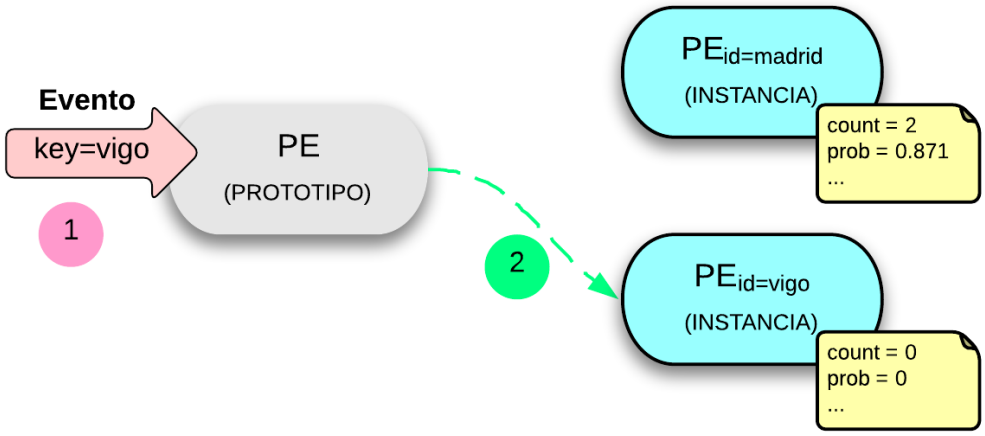

Ilustración 4. Funcionamiento de PEs ante eventos (4/4)

1) Llega un nuevo evento con una clave nueva
2) La aplicación detecta que no existe ninguna instancia del PE destino identificada con dicha clave, por lo que crea una nueva a partir del prototipo. Esta nueva instancia será la encargada de procesar el evento.

Un tipo especial de estos PE son los PE sin clave. Estos consumen todos los eventos entrantes con clave nula del flujo que tienen asociado. Típicamente son usados como la puerta de entrada a un clúster S4.

2.4.2.2. Nodos de procesamiento

Los **Nodos de Procesamiento** (**PN**) son los contenedores o hosts lógicos dentro de un clúster donde se despliegan las aplicaciones, y por tanto donde residen los PE. También se les conoce como particiones del clúster.

Los PN son los responsables de escuchar eventos, ejecutar operaciones ante la llegada de los mismos, y enrutarlos a otros PN con la asistencia de la capa de comunicación.

Todos estos elementos se detallan en la siguiente ilustración [15]:

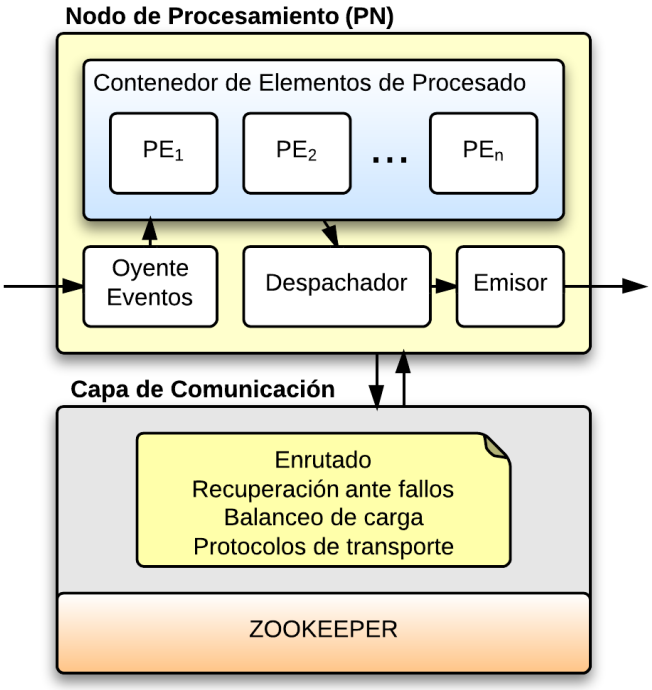

Ilustración 5. Nodo de procesamiento y Capa de comunicación

Cuando se envía un evento por un *stream*, un componente de S4 (el despachador) será el encargado de enrutarlo a un PN determinado, aplicando una función *hash* a la clave del evento. De esta forma S4 proporciona balanceo de carga estático entre las particiones del clúster.

Como ya hemos comentado anteriormente existe un tipo especial de objeto PE: el prototipo del PE, que tiene los 3 primeros componentes de toda instancia de PE, pero el valor de su clave aparece vacío. Este objeto se inicializa en el arranque de cada PN.

Un oyente de eventos en el PN pasa los eventos entrantes al Contenedor de Elementos de Procesado, que es el encargado de invocar los PE adecuados en el orden apropiado. Dependiendo del flujo de llegada y del tipo de PE de destino se ejecutará una de las siguientes opciones teniendo en cuenta la clave del evento:

- Si el evento es sin clave:
  - Si no existe la única instancia posible del PE destino, el PN creará una nueva a partir del prototipo del PE, y le pasará el evento.
  - Si ya existe la instancia del PE, el PN le pasará el evento sin más.
- Si el evento tiene clave:
  - Si no existe la instancia del PE asociado a dicha clave, el PN creará una nueva, y le pasará el evento.
  - Si ya existe la instancia del PE asociado a la clave, el PN le pasará el evento sin más.

Como consecuencia de este diseño, todos los eventos con clave tendrán garantías de enrutarse al PN correspondiente, y dentro del mismo, a la instancia de PE asignada, que será única dentro del PN. Se asegura que un PE con clave sólo esté mapeado a un PN, gracias a la función *hash* aplicada por el despachador de S4. Los PE sin clave en principio pueden ser instanciados en cada PN.

2.4.2.3. Capa de comunicación

La capa de comunicación proporciona la gestión del clúster, el mapeado de los nodos físicos a los lógicos, y la recuperación automática ante errores (detecta fallos de hardware y actualiza dicho mapeo automáticamente). Todo esto lo hace con la ayuda de Zookeeper; el servicio distribuido de coordinación para aplicaciones distribuidas de código abierto que también forma parte de la incubadora de Apache. Al tratarse éste de un servicio distribuido, siempre y cuando lo despleguemos en más de una máquina, la plataforma S4 estará dotada de alta disponibilidad, ya que se evita la existencia de un solo punto de fallo.

Los emisores de eventos especifican sólo nodos lógicos cuando envían mensajes; los nodos físicos son transparentes para ellos, así como el re-mapeo debido a fallos.

La API de esta capa proporciona recubrimientos en varios lenguajes (Java, C++, etc), por lo que tenemos la posibilidad de programar el *driver* en el lenguaje que prefiramos, aunque el núcleo de S4 esté escrito en Java. Podemos usar esta API por ejemplo para enviar eventos de entrada a nodos en un clúster S4.

Cabe señalar que S4 en su última versión usa exclusivamente TCP, con el fin de garantizar la entrega de mensajes a nivel de transporte.

Vemos a continuación un ejemplo de aplicación, donde se aplica todo lo visto hasta ahora y se observa el funcionamiento básico de S4.

2.4.3. Ejemplo de aplicación S4: Twitter *trending topics*

Los desarrolladores de S4 proporcionan con el código fuente de la plataforma una aplicación de ejemplo para probar su funcionamiento (usando Twitter como fuente de datos), lo que supone nuestro primer contacto con S4 y nos proporciona una visión global de la herramienta y de su utilidad.

La aplicación se divide en dos:

- Twitter Adapter (test-apps/twitter-adapter/): adaptador que consume el *feed* de Twitter, convierte los *tweets* (mensajes) en eventos propios de S4 y los manda por un *stream*. Este elemento adaptador aparece por norma general en todas las aplicaciones S4 debido a la necesidad de procesar los datos de entrada con el fin de manejarlos en el resto de la aplicación.
- Twitter Counter (test-apps/twitter-counter/): extrae *hashtags* (temas marcados con el símbolo #) del texto contenido en los eventos S4 que le llegan, y mantiene la cuenta de los mismos, volcando la lista de los más populares periódicamente a disco (lo que se conoce como *trending topics*).

A continuación se muestra el esquema que refleja los elementos de procesado y *streams* de eventos presentes en la aplicación:

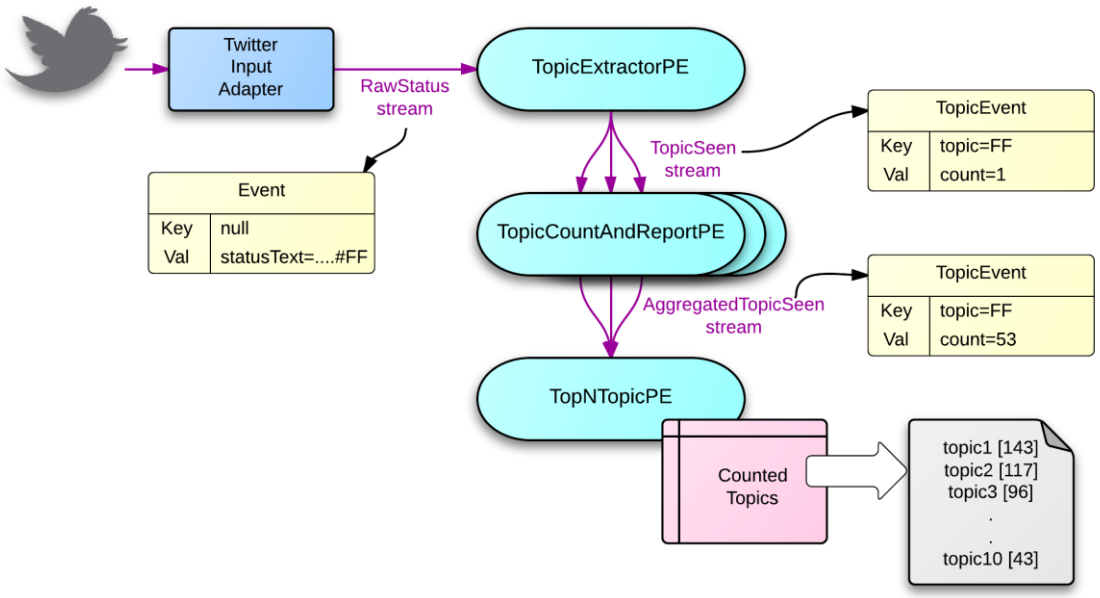

Ilustración 6. Esquema de la Aplicación Ejemplo de Twitter

Modelamos un *stream* en S4 como una secuencia de eventos de la forma (K, D); donde K es la clave y D son los datos. Cada uno de estos dos campos es una tupla identificador-valor.

En el ejemplo descrito en la ilustración 2, los eventos de entrada contienen el texto de un *tweet* como dato a procesar. El propósito final del sistema es producir continuamente una lista ordenada de los 10 temas más comentados en el conjunto de todos los *tweets* procesados, con el mínimo retardo posible.

Los eventos de entrada llegan al sistema S4 sin clave. Un único objeto de la clase TopicExtractorPE escucha todos los eventos de este tipo y procesa los datos. Por cada *hashtag* (tema indicado en Twitter con una almohadilla) presente en el texto, el PE

creará y emitirá un nuevo evento al *stream* TopicSeen de tipo TopicEvent, cuya clave será *topic*=<palabra> y su contenido count=1.

Por cada valor de la clave en los eventos del *stream* TopicSeen, S4 se encarga de buscar una instancia del PE destino, TopicCountAndReportPE, asociada a dicha clave. Si existe, ese objeto PE recibe y procesa el evento, incrementando su contador propio. En caso contrario, se instancia un nuevo objeto TopicCountAndReportPE a partir del prototipo del PE, y se procesa el evento de igual forma.

Cada cierto tiempo, el objeto TopicCountAndReportPE manda un evento de tipo TopicEvent al *stream* AggregatedTopicSeen, con clave *topic*=<palabra> y valor el de su propio contador. Dichos eventos son procesados por una única instancia del objeto TopicNTopicPE, y actualiza la cuenta para dicho *hashtag*. Cada 10 segundos guarda en un fichero la lista con los 10 temas más comentados y su correspondiente número de apariciones.

El despliegue y la ejecución de la aplicación son bastante sencillos. Puede verse detallado en el anexo B.4.

2.4.4. Tolerancia a fallos

Convivir con un sistema distribuido durante un periodo de tiempo continuado, implica la posibilidad de enfrentarse a las siguientes situaciones:

- Fallos en el sistema
- Actualizaciones de la infraestructura
- Reinicios programados
- Redespliegue de aplicaciones

En cada uno de estos supuestos, algún nodo de S4 (si no todos) quedará temporalmente fuera de servicio. El sistema por tanto estará parcialmente no disponible, y el estado en memoria acumulado durante la ejecución podría perderse.

Para lidiar con este problema S4 proporciona:

- Mecanismo de respuesta ante errores, que proporciona alta disponibilidad.
- Recuperación de estado basada en *checkpoints* o puntos de respaldo.

Ambas características respetan la baja latencia de S4 en procesamiento.

2.4.4.1. Mecanismo de respuesta ante errores

Al definir un clúster en S4 se indica el número de tareas o particiones que tendrá de capacidad (máximo de nodos activos a la vez). Un subconjunto de nodos es asignado a las particiones del clúster y pasan a estar activos. El resto de nodos que no consigan hacerse con una partición del clúster, son registrados como nodos en espera para los demás nodos activos.

Para garantizar la disponibilidad ante fallos repentinos en los nodos, S4 proporciona un mecanismo que detecta automáticamente un nodo caído, y redirige los eventos que iban a ese nodo hacia un nodo en espera.

El mecanismo se detalla en las siguientes ilustraciones [39]:

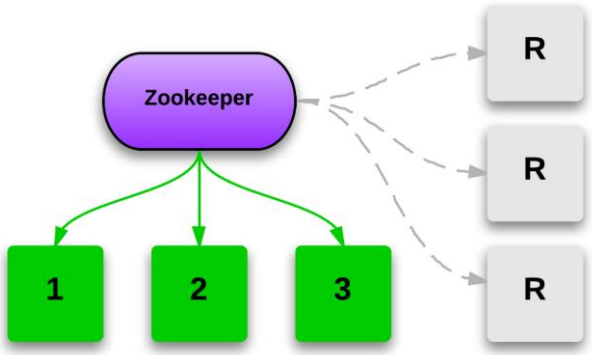

Ilustración 7. Estado inicial

En este ejemplo tenemos 3 particiones disponibles en el clúster para albergar la aplicación, y existen 6 nodos arrancados y registrados por Zookeeper. De estos 6, habrá 3 que se hagan con una partición, y serán los nodos activos. El resto formará el conjunto de nodos en espera. Los nodos activos recibirán mensajes de las particiones constantemente para revisar su estado.

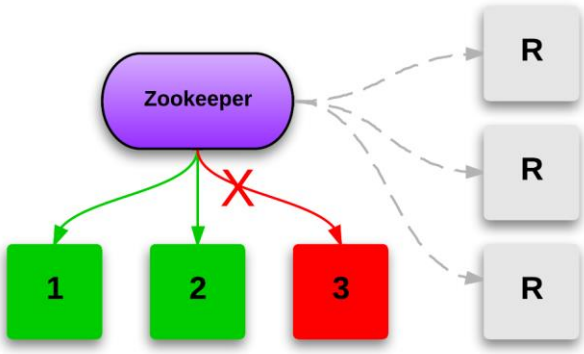

Ilustración 8. Detección de nodo caído

Zookeeper detecta rápidamente errores en un nodo y notifica a los demás nodos. La plataforma considera que un nodo está caído cuando no puede llegar a él tras un *timeout* en una sesión (tiempo especificado por el cliente; como mínimo el doble del denominado *tick time*; parámetro configurado en Zookeeper).

En este ejemplo, falla el nodo de la partición 3 y se notifica a los que ocupan las particiones 1 y 2, así como a los demás nodos en espera.

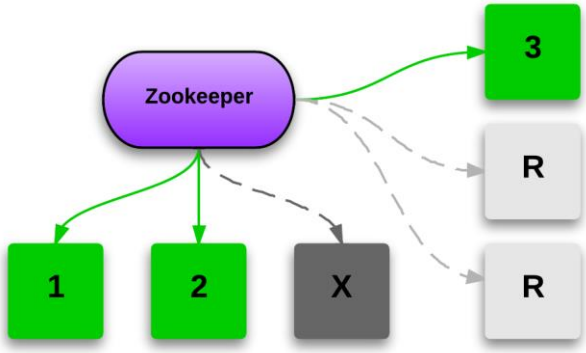

Ilustración 9. Reajuste

Los nodos en espera compiten por la partición disponible, y el primero que responda a Zookeeper se hace con ella. Los demás nodos activos son notificados de esta asignación y pueden enrutar nuevamente sus mensajes a la partición 3.

Esta técnica proporciona alta disponibilidad, pero no previene la pérdida de datos en los nodos caídos. No obstante, como veremos a continuación, S4 cuenta con un mecanismo adicional para acotar el problema de pérdidas al intervalo de tiempo entre la caída de un nodo y la asignación del nuevo (los únicos datos que se perderán serán los de aquellos eventos que lleguen en ese breve espacio de tiempo y que tenga como destino un PE en el nodo con problemas).

2.4.4.2. Recuperación mediante checkpoints

Un *checkpoint* o punto de recuperación, es una copia de seguridad del estado de un objeto PE. El procedimiento consiste en serializar dicho objeto, y almacenar el resultado en un lugar fuera del nodo S4 y accesible desde todas las máquinas; por ejemplo en un NAS (almacenamiento en red) o un directorio compartido.

En caso de fallo en un nodo, este se sustituirá por otro en espera. Este no mantendrá las instancias de los PE que tenía su antecesor, pero si están activados los *checkpoints* será posible recuperar el estado de dichas instancias en el momento de volver a crearlas. Cuando llegue un nuevo evento y sea necesario instanciar un PE concreto, S4 buscará entre los *checkpoints* almacenados si existe uno de dicho objeto, y de ser así lo

deserializará para copiar los campos del objeto guardado al recién creado antes de procesar el evento. La pérdida de estado será mínima.

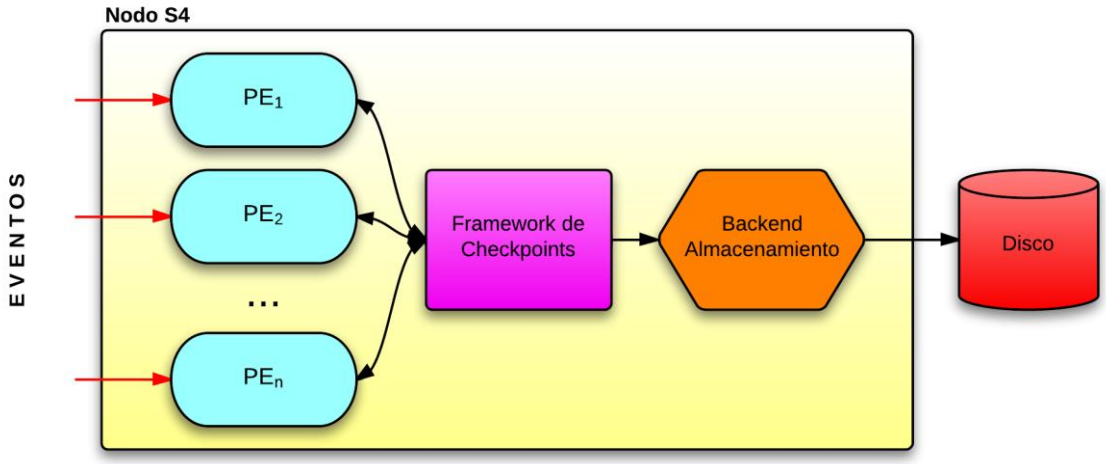

Ilustración 10. Checkpoints en un nodo

Se pueden definir y configurar *checkpoints* de forma particular en cada PE de la aplicación. Estos pueden ser de dos tipos:

- Periódico: el *checkpoint* se hace cada cierto intervalo de tiempo, determinado en la propia aplicación.
- Eventual: el *checkpoint* se lleva a cabo cuando se ha procesado un número concreto de eventos en el PE.

Para minimizar el retardo, los *checkpoints* se harán de forma no coordinada y asíncrona; molestando lo mínimo posible ante la llegada de nuevos eventos.

2.4.5. Procesado de eventos

S4 aplica segmentación en el procesamiento de eventos mediante una serie de **ejecutores** (objetos que ejecutan tareas). Un ejecutor normalmente mantiene una cola limitada de tareas, a ejecutar por un conjunto de hilos. Cuando la cola se llena, el ejecutor sigue un comportamiento u otro dependiendo de la configuración de S4:

- Bloqueo: el hilo actual se bloquea hasta que la cola deje de estar llena. Implementado con semáforos.
- Descarte: la nueva tarea es rechazada.
- Regulación: se limita la tasa máxima de procesamiento; evitando que se envíen mensajes con demasiada frecuencia. Cuando se supera dicha tasa, las nuevas tareas se rechazan.

En la siguiente ilustración se presenta la segmentación en ejecutores al detalle:

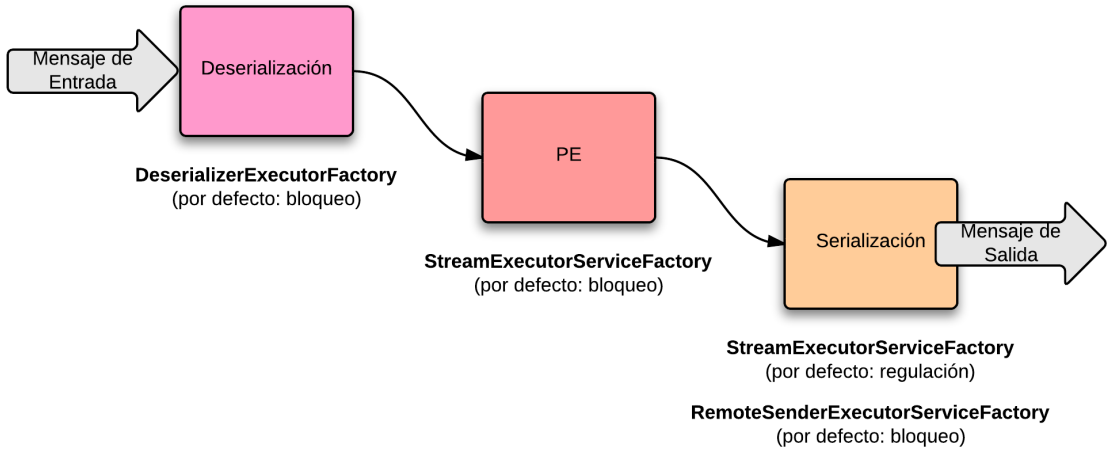

Ilustración 11. Ejecutores y políticas de mantenimiento de colas

Proceso de recepción

1. Se recibe un mensaje a través del *socket* del nodo, en forma de array de bytes.
2. El mensaje se pasa al ejecutor de deserialización, que tiene acceso a las clases de la aplicación en el nodo, y por tanto sabe cómo convertir ese mensaje en un objeto (evento). Por defecto este ejecutor usa 1 hilo, y bloquea las nuevas tareas si su cola de entrada se llena.
3. El evento se pasa al ejecutor de un *stream* concreto, seleccionado en función de la información que contiene el evento. Por defecto bloquea las nuevas tareas si la cola de procesamiento se llena.
4. El evento se procesa en la instancia del PE que corresponde dependiendo de la clave del evento.

Proceso de emisión

1. El PE inyecta un evento en su *stream* de salida.
2. El evento se pasa a un ejecutor u otro dependiendo del destino:
    - Si es para otro clúster, el evento va a un ejecutor de envíos remotos
    - Si es para el mismo, hay dos opciones:
        a) Que vaya al mismo nodo, en cuyo caso se pasa directamente al ejecutor del *stream* (punto 3 de recepción).
        b) Que vaya a un nodo distinto, pasando entonces a un ejecutor de envíos locales.
3. El ejecutor de envíos locales/remotos serializa el evento convirtiéndolo en un array de bytes. Dependiendo del tipo, su comportamiento al llenarse su cola de procesamiento será distinto por defecto:
    - El ejecutor de envíos remotos bloquea las tareas nuevas.

- El ejecutor de envíos locales usa regulación, con una tasa máxima configurable. Si los eventos llegan con una frecuencia más alta, se descartan.

Capítulo 3.
Diseño

Una vez elegidas las bases para nuestra aplicación (el algoritmo Naïve Bayes y la plataforma S4) es posible pasar a la siguiente etapa.

El desarrollo de una aplicación S4 comienza por una fase de diseño preliminar a la implementación, con el fin de definir claramente la estructura necesaria para procesar satisfactoriamente los datos de entrada a lo largo de varias etapas.

Las aplicaciones S4 pueden ser siempre vistas como un grafo conformado por dos elementos básicos:

- Elementos de procesado (vértices)
- *Streams* de eventos que interconectan esos elementos (aristas)

Por normal general las aplicaciones S4 cuentan con un elemento especial ajeno a la propia aplicación denominado adaptador, que convierte los datos de la fuente en eventos. Estos eventos serán la entrada propiamente dicha de la aplicación.

Una aplicación S4 cuenta siempre con un PE de entrada, que será el encargado de recibir todos los eventos emitidos por el adaptador. En realidad S4 concede al desarrollador la oportunidad de omitir el adaptador y conectar el PE de entrada directamente al *stream* proveniente de la fuente de datos, pero a la vez se recomienda encarecidamente el uso del adaptador porque aporta varias ventajas; entre las que destacan el procesado previo de los datos fuera de la aplicación, y la posibilidad de escalar ese procesado, ya que el adaptador también es una aplicación S4.

Para una aplicación que realmente aproveche la funcionalidad y ventajas que aporta S4, existirán varios PE adicionales para el procesamiento de los datos. Por ejemplo, es típico el uso de un PE para procesar los eventos emitidos por el de entrada a modo de función *map*; y también se suele usar otro PE adicional para recopilar y procesar resultados intermedios sacando conclusiones finales, a modo de función *reduce*. Podremos añadir siempre más tipos de PE según nuestras necesidades, pero es recomendable encontrar un equilibrio en la fase de diseño para evitar etapas de más en la aplicación final que podrían repercutir en el rendimiento.

En las manos del desarrollador queda también el elegir si un PE podrá tener una o más instancias. Por ejemplo el PE de entrada tendrá una única instancia, al igual que un típico PE encargado de recopilar resultados; pero si queremos procesar en paralelo por ejemplo las distintas palabras de las que se compone el texto contenido en un evento, el PE encargado de esa tarea deberá contar con múltiples instancias. La posibilidad de que un PE pueda tener varias o una única instancia no es una propiedad que se implemente en la clase correspondiente del PE, sino que viene dado por la implementación de la propia aplicación o simplemente por la clave de los eventos que le lleguen al prototipo del PE en tiempo real (si siempre enviamos eventos a un prototipo de PE con la misma clave, sólo existirá una instancia del mismo).

Hay que tener siempre en cuenta que los objetos PE envían eventos de forma asíncrona (cuando sea necesario) y que estos son despachados a otro objeto PE dependiendo del *stream* donde se inyecte el evento, y de la clave del mismo.

Una buena forma de empezar a realizar el diseño podría ser la recopilación de determinada información. Estos son los pasos que se han seguido en nuestro caso:

- Disgregar funcionalidad por etapas para obtener los distintos tipos de PE necesarios y los *streams* que los comunican.
- Fijar la clave de los eventos que se intercambiarán y tener en cuenta si serán necesarias una o más instancias del PE de destino.
- Analizar los datos que se enviarán en los eventos de cada *stream*.
- Aclarar los datos que se quieren obtener y la forma en que se guardarán (ficheros, base de datos, …)
- Analizar las propiedades que necesitará cada tipo de PE para guardar datos locales (estado interno), y apuntar las que se quieren proteger en caso de errores, de cara a utilizar el mecanismo de *checkpoints*. Las propiedades necesarias de conservar tendrán que ser serializables, y no estar definidas con la palabra clave *transient* (que es lo que identificará a las propiedades que no queramos guardar).

Diseñamos nuestra máquina desambiguadora con aprendizaje automático valiéndonos de la documentación y ejemplos de S4, así como de las necesidades de la misma.

A continuación se muestra el esquema de la aplicación final, teniendo en cuenta los PE necesarios, los *streams* de eventos, la estructura de los mismos, los datos almacenados en los PE, etc.

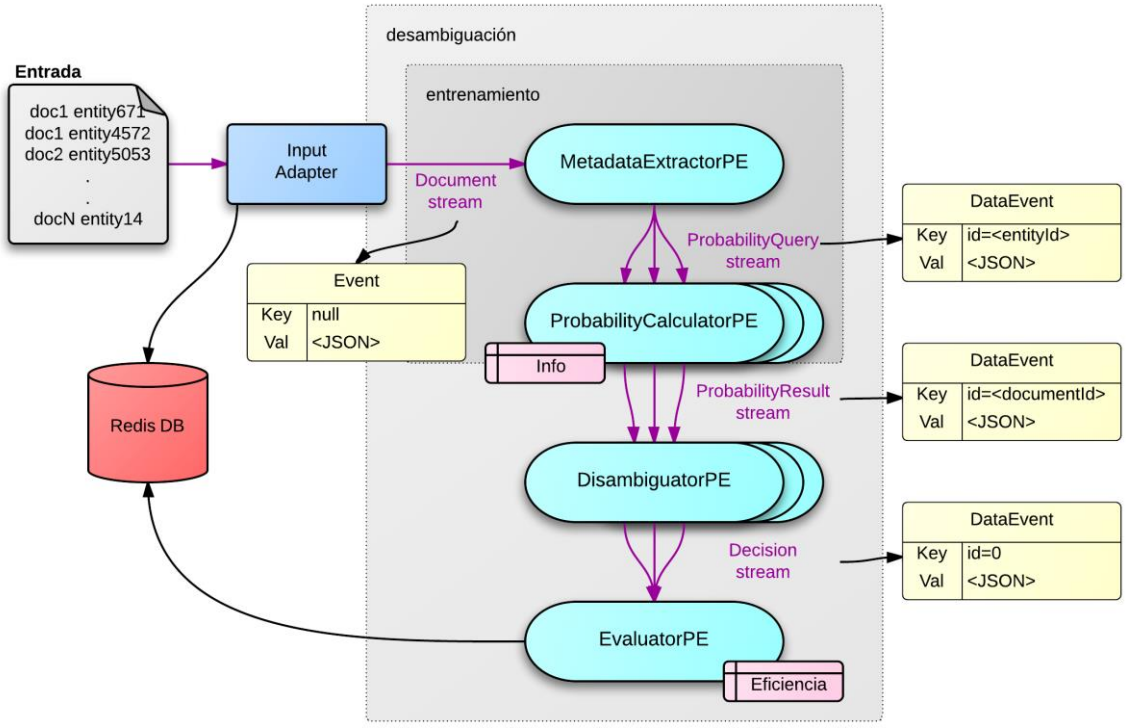

Ilustración 12. Esquema de la aplicación Desambiguador

La funcionalidad de los distintos elementos presentes en el esquema de la aplicación es la siguiente:

- **InputAdapter**: Adaptador de la entrada (instancia única). Convierte los artículos periodísticos de la fuente de datos en eventos que sabe interpretar el PE de entrada de nuestra aplicación. El contenido de dichos eventos se explica más adelante.
- **MetadataExtractorPE**: PE de entrada de la aplicación (instancia única). Procesa los eventos de entrada al sistema provenientes del adaptador para extraer las entidades del artículo correspondiente. Analiza si la muestra es para desambiguación (es necesario resolver una entidad ambigua) o aprendizaje (todas las entidades son conocidas). Emite eventos para el siguiente tipo de PE; uno por entidad bajo estudio.
- **ProbabilityCalculatorPE**: Guarda estadísticas en modo aprendizaje y las usa en modo desambiguación con el fin de estimar la probabilidad de que una

mención pertenezca a una entidad candidata (una instancia por entidad). En este último modo envía un evento con el resultado al siguiente PE.

- **DisambiguatorPE**: En base a las probabilidades estimadas por distintas instancias del anterior PE, elige una de las entidades candidatas como aquella que con más probabilidad resuelve la ambigüedad (con una instancia por desambiguación).
- **EvaluatorPE**: Si para entrenamiento contamos con un corpus de datos en el que previamente se han resuelto todas las ambigüedades (por ejemplo, gracias a un editor) en lugar de tener un *stream* con los artículos en bruto, este PE evaluará la eficiencia de nuestro desambiguador (con una única instancia). Dicha eficiencia será mostrada cada vez que se realice una nueva desambiguación.

A continuación se detalla el diseño del conjunto dividido por componentes.

3.1. ENTRADA Y ADAPTADOR

Al no contar en la práctica con un flujo de datos en tiempo real para probar nuestra aplicación, haremos las pruebas de sistema utilizando un corpus, que contendrá la información relevante de un largo número de documentos extraídos del New York Times. Dicho corpus cuenta con 1.478.767 artículos, abarcando unos 20 años de noticias del periódico.

El corpus será procesado por un adaptador implementado a medida (InputAdapter), y la información de los documentos será servida a la entrada de nuestra aplicación (MetadataExtractorPE), con un tiempo entre llegadas que sigue una distribución determinada para simular el flujo en tiempo real (para la implementación de la aplicación el origen de los datos es transparente).

Por cada artículo lo que tenemos realmente es un conjunto de metadatos ya extraídos. Cada línea del corpus pertenece a un metadato o entidad de un artículo determinado, publicado en una fecha concreta:

```
<ruta del artículo>  <id de entidad>   <día de publicación>
```

Un ejemplo del contenido del corpus:

```
0000041.xml brauen_fred.per 1987-01-01
0000041.xml central_park_-nyc-.loc  1987-01-01
0000041.xml new_york_city.loc       1987-01-01
```

```
0000041.xml transportation_authority.org 1987-01-01

0000042.xml intriligator_m_d.per    1987-01-01
0000042.xml iran.loc        1987-01-01
0000042.xml national_security_council.org   1987-01-01
0000042.xml nicaragua.loc   1987-01-01


0000043.xml traffic_safety_administration.org      1987-01-01
0000043.xml steed_diane_k.per       1987-01-01
0000043.xml transportation_department_of_-us-.org   1987-01-01
0000043.xml united_states.loc       1987-01-01
```

El adaptador agrupará los metadatos de un mismo artículo, y generará la entrada para la aplicación en un lenguaje estructurado; en nuestro caso JSON [41]; información que irá incluida en un evento de S4 sin clave.

La aplicación está preparada para aceptar entidades que pueden representar a personas, lugares, asociaciones, organismos, etc. Por ejemplo: “united_states.loc” o “rafael_nadal.per”.

En los datos originales del corpus no aparecen ambigüedades para poder evaluar nuestra aplicación, por lo que será el propio adaptador quien las genere a partir de los datos originales, y de esa forma podremos conocer la efectividad de nuestras estimaciones. La forma de llevar a cabo esto es definiendo dos entidades en configuración que serán sustituidas por un seudónimo ambiguo cada vez que una de ellas aparezca entre los metadatos del artículo de entrada; esta técnica se denomina *pseudo-names* [23] y su uso es bastante común a la hora de evaluar mecanismos de desambiguación.

El procedimiento se detalla en la siguiente ilustración; la ambigüedad se genera cuando aparece en un artículo la entidad fernando_alonso.per o la entidad xabi_alonso.per, y se manda un evento para desambiguación con la entidad ambigua:

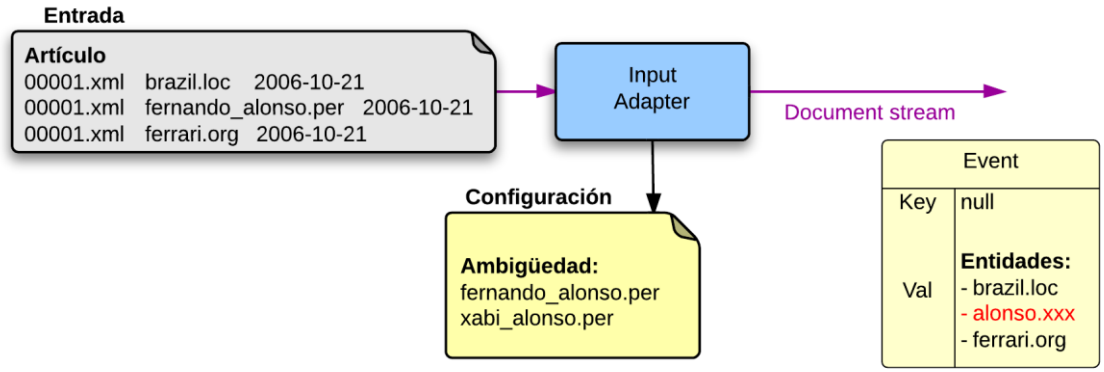

Ilustración 13. Técnica de pseudo-names usada al evaluar la aplicación

Si por un casual aparecieran las dos entidades configuradas en un mismo artículo, se descartará sin más, ya que no aporta nada a la evaluación.

Los datos originales del artículo se guardan en una base de datos Redis [42], de forma que el evaluador final pueda consultar si el desambiguador funcionó bien o por el contrario se equivocó en su decisión.

Un escenario real de resolución de entidades, podría ser el caso de un editor que se dedica a desambiguar entidades en las noticias de un medio de comunicación. Como es natural, en este escenario los artículos van intercalados con las correcciones. En el caso de nuestra aplicación, lo que hará el adaptador es enviar primero un artículo en el que nuestra aplicación encontrará un seudónimo ambiguo; y acto seguido mandará el artículo original del corpus con todas las entidades resueltas. El comportamiento de la aplicación ante el primer evento es lo que se denomina en esta memoria el modo de desambiguación; mientras que el comportamiento ante el segundo evento se denomina modo de aprendizaje o entrenamiento.

A continuación se detallan ambos modos, siempre con la mente puesta en la implementación del algoritmo de Naïve Bayes. Se expone en primer lugar el funcionamiento del aprendizaje, a pesar de ocurrir en nuestro caso después de la desambiguación; esto es debido a que resulta conveniente conocer la información que se almacena al procesar una muestra de entrenamiento previamente a explicar cómo se utiliza para llevar a cabo una desambiguación.

3.2. APRENDIZAJE

Se considera modo de entrenamiento o aprendizaje aquél en que se procesa un documento sin entidades ambiguas, con el fin de poder desambiguar mejor en un futuro inmediato. Será el nodo de entrada a nuestro sistema quien determine el modo de trabajo de la aplicación para cada evento recibido.

El evento que llega al **MetadataExtractorPE**, contiene la siguiente información (siendo "id" la clave y "data" los datos):

```
id: null
data:
{
    "documentId"=<string>,
    "context"={
       "entities": [<string>]
    },
    "doubt"=false
}
```

La clave será nula, como corresponde a los eventos de entrada a un sistema S4. Los datos en formato JSON incluyen:

- *documentId*: identificador unívoco del documento.
- *context*: metadatos que definen el contexto y que servirán a nuestra aplicación para subsecuentes peticiones de desambiguación. En nuestro caso existe un único tipo aceptado:
  - *entities*: lista de entidades.
- *doubt*: booleano igual a *false* en aprendizaje.

Ejemplo:

```
id: null
data:
{
    "documentId"="file23523",
    "context"={
       "entities": ["madrid.loc", "mostoles.loc"]
    },
    "doubt"=false
}
```

El PE actualizará una variable interna con el número de documentos procesados en aprendizaje; valor necesario para aplicar Naïve Bayes:

```
trainingDocs ++;
```

Se generarán nuevos eventos para ser enviados al siguiente elemento de procesado (**ProbabilityCalculatorPE**); tantos como entidades albergue el documento original. Dichos eventos tienen esta forma:

```
id: <string>
data:
{
    "documentId"=<string>,
    "context"={
       "entities": [<string>]
    },
    "queries"=0,
    "totalCandidates"=0,
    "totalEntities"=0
    "trainingDocs"=0
}
```

La clave será el identificador de la entidad protagonista del evento. Los datos en formato JSON del evento incluyen:

- *documentId*: cadena vacía en aprendizaje.
- *context*: conjunto de las demás entidades, excluyendo a la que va como clave del evento.
- *totalCandidates*: 0 en aprendizaje
- *totalEntities*: 0 en aprendizaje
- *trainingDocs*: 0 en aprendizaje

Para nuestro ejemplo, se enviarán 2 eventos, que irán a instancias distintas del ProbabilityCalculatorPE:

```
id: "madrid.loc"
data:
{
    "documentId"="",
    "context"={
       "entities": ["mostoles.loc"]
    },
    "totalCandidates"=0,
    "totalEntities"=0,
    "trainingDocs"=0
}
```

```
id: "mostoles.loc"
data:
{
    "documentId"="",
    "context"={
       "entities": ["madrid.loc"]
    },
    "totalCandidates"=0,
    "totalEntities"=0,
    "trainingDocs"=0
}
```

En la instancia que atiende eventos con clave "madrid.loc", se incrementarán los siguientes valores guardados en memoria y necesarios para desambiguación mediante Naïve Bayes:

- Nº apariciones de "madrid.loc"
- Nº apariciones conjuntas de "madrid.loc" y "mostoles.loc"

El proceso se hará de forma análoga en cada instancia a la que llegue un evento, actualizando sus contadores internos correspondientes.

Aquí terminaría el procesado de nuestra aplicación ante muestras de aprendizaje. Vemos a continuación el comportamiento ante muestras para desambiguación.

3.3. DESAMBIGUACIÓN

Se considera modo de desambiguación o clasificación aquél en que se procesa un documento con una mención ambigua o desconocida, que es necesario asociar a una entidad. En este modo, nuestra aplicación tendrá que poner en práctica todo lo aprendido con los documentos procesados en entrenamiento, e intentará estimar por medio del algoritmo Naïve Bayes a qué entidad de las dos configuradas es más probable que haga referencia la mención.

El evento que llega al primer elemento de procesado (MetadataExtractorPE) contiene la siguiente información:

```
id: null
data:
{
    “documentId”: <string>,
    "context": {
        "entities": [<string>]
    },
    "doubt": true
}
```

La clave será nula, como corresponde a todos los eventos de entrada. Los datos en formato JSON del evento incluyen:

- *documentId*: identificador del documento.
- *context*: entidades conocidas que aparecen junto a la mención ambigua.
- *doubt*: booleano con valor *true*.

Se actualizará una variable interna con el número de peticiones recibidas de desambiguación:

```
disambiguationQueries ++.
```

Se generarán nuevos eventos para ser enviados al siguiente elemento de procesado (ProbabilityCalculatorPE); tantos como entidades candidatas tengamos para deshacer la ambiguedad. Dichos eventos tienen esta forma:

```
id: <string>
data:
{
    "documentId": <string>,
    "context"={
       "entities": [<string>]
    },
    "totalCandidates"=<int>,
    "totalEntities"=<int>
    "trainingDocs"=<int>
}
```

La clave será el identificador de la entidad candidata. Los datos en formato JSON incluyen:

- *documentId*: identificador de documento, para poder reagrupar los resultados de las estimaciones con posterioridad, así como para recuperar los datos originales del corpus con el fin de evaluar la eficiencia.
- *context*: conjunto de entidades, excluyendo a la candidata.
- *totalCandidates*: número total de entidades candidatas; en nuestro caso 2.
- *totalEntities*: número total de entidades aparecidas hasta el momento en entrenamiento.
- *trainingDocs*: número de documentos procesados hasta el momento en el modo de entrenamiento.

En cada instancia de ProbabilityCalculatorPE a la que llega un evento con una petición de desambiguación lo que se hace es aplicar el algoritmo Naïve Bayes (ver apartado 2.2) para estimar la probabilidad de que la entidad desconocida sea la que aparece como clave en los eventos que procesa. Para ello, además de usar los valores de *totalEntities* y *trainingDocs*, consultará en memoria las estadísticas de apariciones de la entidad candidata en solitario o en conjunto con el resto del contexto; todo ello proveniente de artículos procesados anteriormente como entrenamiento.

En nuestro ejemplo, estas serán las estadísticas a tener en cuenta para la instancia de ProbabilityCalculatorPE que procesa eventos con clave “madrid.loc”:

- Nº apariciones de “madrid.loc”
- Nº apariciones conjuntas (de “madrid.loc”) con “mostoles.loc”

Estos datos están almacenados en una estructura *hashmap*, de forma que cada elemento del contexto cruzado con “madrid.loc” sea la clave de la entrada, o la propia “madrid.loc” para el caso de las apariciones de dicha entidad.

Cada instancia de ProbabilityCalculatorPE se encarga de estimar la probabilidad de su entidad candidata como ya hemos indicado siguiendo el algoritmo Naïve Bayes. El

resultado será enviado de nuevo a otro PE (**DisambiguatorPE**) encargado de esperar el resto de estimaciones, y decidir finalmente qué entidad es la más probable de entre todas las candidatas. Dichos eventos tienen la forma:

```
id: <string>
data:
{
    "candidateId"=<string>,
    "probability"=<double>,
    "totalCandidates"=<int>
}
```

La clave será el identificador del documento. Los datos en formato JSON incluyen:

- *candidateId*: identificador de la entidad candidata.
- *probability*: probabilidad estimada de que la entidad candidata sea el significado correcto para la mención.
- *totalCandidates*: número total de entidades candidatas.

Cada instancia de este PE agrupará los resultados recibidos para las estimaciones asociadas a un mismo documento, y decidirá sobre la entidad que con más probabilidad resuelve la ambigüedad.

Si existe un empate entre los resultados de las probabilidades estimadas, debido a que ambas entidades candidatas han aparecido las mismas veces y el contexto es nuevo (por ejemplo cuando no existen precedentes) se escogerá una entidad al azar.

Dicho resultado será enviado a un último elemento de procesado (**EvaluatorPE**), encargado de evaluar si la decisión del sistema ha sido correcta o no. Dichos eventos tienen la forma:

```
id: "result"
data:
{
    "documentId"=<string>,
    "resultId"=<string>,
}
```

La clave es genérica y arbitraria ya que sólo necesitamos una instancia de este PE. Los datos en formato JSON incluyen:

- *documentId*: identificador del documento.
- *resultId*: identificador de la entidad elegida para solventar la desambiguación.

Como en nuestro caso utilizamos un corpus con ambigüedades forzadas, podremos consultar si el resultado de nuestra estimación es correcto o no comparando con los

datos originales del documento; información que el adaptador guardó previamente en Redis. En caso de acierto incrementaremos una variable interna; teniendo así en todo momento la estadística de efectividad de nuestro desambiguador.

Inmediatamente después de esto, metemos de nuevo el artículo a nuestro sistema, esta vez sin entidades ambiguas, para que nuestra máquina siga aprendiendo.

En el siguiente capítulo se detalla la implementación de nuestra aplicación a partir de este diseño previo y teniendo en cuenta las particularidades de la plataforma S4.

Capítulo 4. Implementación

En los capítulos anteriores hemos podido ver a grandes rasgos la teoría del algoritmo Naïve Bayes y el funcionamiento básico de S4; bases de nuestra aplicación. También hemos realizado el diseño de la misma teniendo en cuenta tanto los requisitos como dichas bases. En el presente capítulo se detalla la implementación llevada a cabo, comenzando por la instalación de S4, ya que se hace uso de una funcionalidad específica para crear el esqueleto de una aplicación.

Aparte de todo lo que se explica en esta memoria, siempre está a disposición de los desarrolladores el *javadoc* de la API de S4 [43], que servirá como referencia principal para ver cómo funcionan todas las clases y métodos al detalle.

4.1. INSTALACIÓN DE S4

La descarga de S4 se realiza desde el espacio web que tiene la aplicación dentro de la incubadora de proyectos de Apache [39], o también desde su repositorio Git [44]. La versión estable en el momento de elaboración de este estudio era la 0.6.0 final (en las primeras fases se trabajó con la versión 0.5.0, pero se dio el salto cuando apareció la versión 0.6.0 estable, debido a las mejoras de rendimiento incluidas y corrección de bugs).

Se recomienda siempre elegir la versión que incluye el código fuente en vez de la binaria, por tema de dependencias. Esto implica que tendremos que hacer un *build* inicial para crear los ficheros binarios.

S4 se ejecuta sobre la máquina virtual de Java, por lo que es necesario tener instalado dicho *software* (ver anexo C.1)

Repetiremos las siguientes acciones en cada servidor donde vayamos a instalar la plataforma S4:

1. Descargar el fichero comprimido con el código fuente desde la página web del proyecto o directamente desde el terminal para ahorrar un paso. Descomprimirlo por medio del comando *unzip* y renombrar el directorio resultante a nuestro gusto:

```
wget http://www.apache.org/dist/incubator/s4/s4-0.6.0-
incubating/apache-s4-0.6.0-incubating-src.zip
unzip apache-s4-0.6.0-incubating-src.zip
mv apache-s4-0.6.0-incubating-src s4-0.6.0
cd s4-0.6.0
```

2. Hacer el *build* del código fuente de S4 usando Gradle [45]. A partir de la versión 0.6.0 de S4, Gradle no viene incluido con el software de la plataforma, por lo que hay que instalarlo en todos los servidores. En el apéndice C.2 se explica al detalle este proceso y lo que se necesita hacer para integrarlo con S4.
   Una vez llevada a cabo la instalación e integración de Gradle, ejecutaremos la siguiente instrucción, que nos construirá los paquetes e instalará los artefactos requeridos en el repositorio Maven [46] local. Se puede consultar el log completo del proceso para obtener más detalle.

```
./gradlew install
```

3. Construir las herramientas de S4, de tal manera que se pueda usar el comando *s4* desde el directorio actual.

```
./gradlew s4-tools:installApp
```

4. Opcionalmente ejecutaremos las pruebas de regresión de S4, aunque no es necesario usando una versión estable:

```
./gradlew test
```

Con esto ya tendremos S4 instalado y listo para crear clústeres y nodos, así como para ejecutar aplicaciones sobre ellos.

Echamos un breve vistazo al directorio principal de S4, que contiene entre otras cosas:

- Ejecutable de S4.
- *Wrapper* y configuración de Gradle.
- Ficheros con información útil.
- Directorio lib/ para librerías y ficheros de propiedades.
- Directorio subprojects/ con los diferentes módulos de los que se compone S4:
  - s4-core
  - s4-base
  - s4-tools
  - s4-comm
  - s4-edsl
  - s4-benchmarks
  - s4-example
- Directorio test-apps/ con algunos ejemplos de utilidad, totalmente funcionales.

Para poder importar proyectos en Eclipse [47] (es altamente recomendable usar un *framework* para desarrollar las aplicaciones) y que las librerías estén disponibles en el *classpath*, ejecutaremos lo siguiente en el directorio base de S4:

```
./gradlew eclipse
```

Si usamos IntellijIDEA [48]:

```
./gradlew idea
```

En el siguiente punto se expone un procedimiento común al desarrollo de cualquier aplicación S4.

4.2. CREACIÓN DEL ESQUELETO DE LA APLICACIÓN

S4 proporciona varios scripts para simplificar el desarrollo y testeo de aplicaciones. Para comenzar a implementar una nueva aplicación se recomienda el uso de una utilidad incluida en S4, que creará un buen punto de partida con una simple instrucción. Gracias a esto obtendremos algo totalmente ejecutable sin necesidad de tocar nada de código.

Lo único que tenemos que hacer es crear un nuevo proyecto usando el comando *newApp*, que nos dará una plantilla básica. Le indicaremos el directorio padre donde se ubicará la aplicación.

```
./s4 newApp disambiguator -parentDir=test-apps

  ______  _  _
 / ____|| || |
| (___  | || |_
 \___ \ |__   _|
 ____) |   | |
|_____/    |_|

 You just created a new S4 project in test-apps/disambiguator!

(...)
```

Podemos revisar la estructura creada dentro del directorio destino:

src/	→ ficheros fuente (siguiendo estructura Maven)
lib/	→ librerías adicionales
build.gradle	→ la plantilla que podremos modificar para el *build*
gradlew	→ referencia al *script gradlew* de la instalación
s4	→ referencia al *script* s4 de la instalación
settings.gradle	→ ajustes de Gradle específicos para este proyecto

Dentro del directorio src/main/java/hello/ podemos encontrar los ficheros HelloPE.java, HelloApp.java y HelloInputAdapter.java. Estos ficheros conforman la aplicación HolaMundo de S4, de la que podemos ver su ejecución en el apéndice B.3. Usaremos estos ficheros como punto de partida para crear nuestras clases, aunque por ejemplo en la implementación final se ha sacado el adaptador como aplicación independiente de cara a tener más flexibilidad en el proceso de despliegue dentro de entornos distribuidos.

En el siguiente punto se explica todo lo referente a la implementación particular de nuestra aplicación.

4.3. DESARROLLO

Existen varios tipos de clases básicas en S4 que debemos conocer a la hora de implementar nuestras aplicaciones:

- **App**: aplicación principal.
- **AdapterApp**: aplicación del adaptador.
- **ProcessingElement**: elemento de procesado.
- **Event**: evento S4.
- **Stream**: *stream* de eventos.

Lo que hará un desarrollador es escribir sus propias clases extendiendo estas (cuando resulte necesario) y sobrescribiendo algunos de sus métodos para conseguir la funcionalidad deseada.

La estructura de paquetes y clases en la aplicación y el adaptador queda de la siguiente manera:

Aplicación

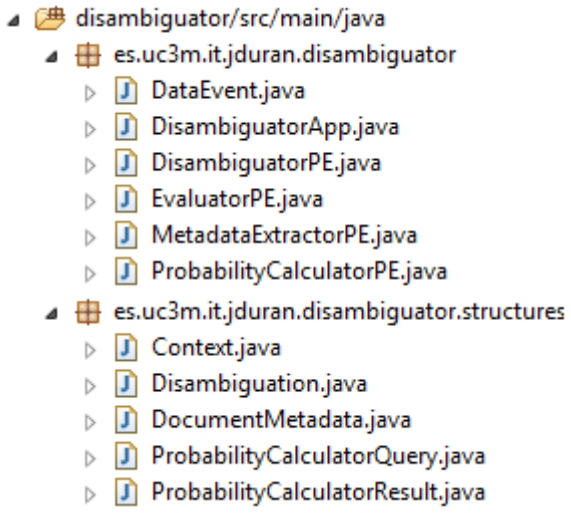

Ilustración 14. Estructura de la aplicación principal

Adaptador

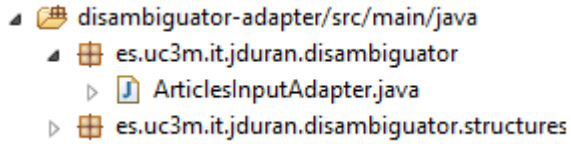

Ilustración 15. Estructura del adaptador

Cabe destacar que el paquete es.uc3m.it.jduran.disambiguator.structures es compartido por aplicación y adaptador; dentro se encuentran las estructuras utilizadas para volcar y manejar los datos de los eventos que circulan por cada *stream*.

A continuación se explican en detalle los elementos claves de la aplicación, los métodos que sobrescriben de las clases a las que extienden, así como otras particularidades.

4.3.1. Adaptador

La clase del adaptador para nuestra aplicación (InputAdapter) extiende a la clase **org.apache.s4.core.adapter.AdapterApp**; que a su vez extiende a otra clase básica de S4: **org.apache.s4.core.App.**

Como ya definimos anteriormente de forma genérica, la funcionalidad básica del adaptador consiste en convertir los datos de la fuente de entrada (en este caso el corpus de artículos del New York Times) en eventos S4.

Además en nuestro caso el adaptador debe crear los seudónimos ambiguos para la aplicación, tarea que llevará a cabo cada vez que encuentre una (y sólo una) de las entidades usadas para crear el seudónimo entre los metadatos de un artículo. Es justo en ese supuesto cuando el adaptador envía un evento por el *stream* de entrada a la aplicación, para que ésta lleve a cabo su desambiguación. Además guarda en base de datos la entidad original para que un elemento de procesado posterior al desambiguador pueda consultarla, y de ese modo evaluar si la decisión fue o no correcta.

Justo a continuación de enviar el evento con el artículo para desambiguación, el adaptador envía otro evento con la entidad original, para que la aplicación pueda aprender.

Estos son los métodos que sobrescribe la clase del adaptador:

- **onInit()** → método ejecutado al inicializar el adaptador. Instancia un hilo que se encargará de enviar la información de los artículos que se vaya añadiendo en una cola.
- **onStart()** → método ejecutado después de la inicialización. Arranca el hilo instanciado en onInit() y llama a un método propio que implementa la funcionalidad principal descrita con anterioridad.

El funcionamiento de la cola, la conversión en eventos y su envío por el *stream* de salida del adaptador se explica a continuación.

La cola es un objeto de la clase java.util.concurrent.LinkedBlockingQueue, donde se añaden elementos procesados mediante el método add().

```
LinkedBlockingQueue<String> messageQueue;

(...)

messageQueue.add(document.toJson());
```

Cada elemento no es más que una cadena de texto en formato JSON con la información extraída de un artículo del corpus de entrada. Esta cadena se extrae de la cola, y se convierte en un evento genérico (objeto de la clase org.apache.s4.base.Event) de la siguiente manera:

```
String jsonArticle = messageQueue.take();
Event event = new Event();
event.put("data", String.class, jsonArticle);
```

Con el método put() de la clase Event, lo que hacemos es incluir información en dicho evento; en este caso la cadena de texto en el campo "data". Como este evento es para la entrada de la aplicación se envía sin clave.

El envío del evento se realiza gracias al método put() de la clase Stream:

```
getRemoteStream().put(event);
```

Con el método getRemoteStream() se obtiene el *stream* de salida, definido en tiempo de despliegue (al crearse el nodo para el adaptador se especifica el nombre del *stream* con la opción –p=s4.adapter.output.stream=<Nombre>)

4.3.2. Elementos de procesado

Los PE que conforman nuestra aplicación son clases que extienden a la clase de S4 **org.apache.s4.core.ProcessingElement.**

Estos son los métodos más importantes que sobrescriben nuestras clases:

- **onCreate()** → método ejecutado cuando se crea una instancia del PE. Usado típicamente para inicializar variables.
- **onEvent(Event event)** → método ejecutado cada vez que llega un evento nuevo a la instancia del PE, de la clase especificada como parámetro. Es aquí donde se lleva a cabo la implementación de la funcionalidad principal del PE.
- **onRemove()** → método ejecutado cuando se elimina una instancia del PE. Usado principalmente para liberar recursos.

Un PE se suscribe a los eventos de un *stream* con una determinada clave o clave nula. Esto se configura en la clase principal de la aplicación, como ya veremos más adelante. El *stream* donde vuelca los eventos de salida también se indica desde la clase principal, invocando el método setDownStream() del PE con dicho *stream* como parámetro:

```
public void setDownStream(Stream<DataEvent> stream) {
    this.downStream = stream;
}
```

Los eventos se emiten de forma análoga a como se hace en el adaptador, mediante el método put(). La clase DataEvent del ejemplo es una personalización de los eventos S4 extendiendo la clase Event. Veremos esto en detalle en el siguiente apartado.

Para extraer la información de un evento de entrada se usa el método get() de la clase Event o un *getter* propio de nuestra clase personalizada. Para adaptar dicha información, se utiliza la librería Gson [49], que nos ayuda a realizar la conversión entre un texto estructurado en formato JSON y una clase propia (en este caso DocumentMetadata) creada para poder manejar los datos de forma más cómoda. He aquí un ejemplo de todo lo anterior:

```
String data = event.get("data", String.class);
DocumentMetadata docMetadata;
docMetadata = gson.fromJson(data, DocumentMetadata.class);
```

4.3.3. Eventos

La clase básica que implementa los eventos en S4 es **org.apache.s4.base.Event**. Para ampliar la funcionalidad de estos eventos bastará con extender dicha clase.

En nuestra aplicación se implementa la clase DataEvent, que no es más que una extensión de la clase Event con dos propiedades de tipo String para guardar clave y datos.

```
public class DataEvent extends Event {

  String id;
  String data;

  public DataEvent(String id, String data) {
      this.id = id;
      this.data = data;
  }
```

```
  public String getId() {
      return id;
  }

  public String getData() {
      return data;
  }

}
```

Esto simplifica aún más el uso de eventos desde los PE.

4.3.4. Clase principal

La clase principal de nuestra aplicación extiende a la clase **org.apache.s4.core.App.**

Existen 3 métodos que se sobrescriben normalmente para extender su funcionalidad:

- **onInit()** → método ejecutado por el contenedor al inicializar la aplicación. Este es el método más importante; aquí se crean y configuran los prototipos de nuestros elementos de procesado, y los *streams* de eventos que los interconectan.
- **onStart()** → método ejecutado por el contenedor después de la inicialización.
- **onClose()** → método ejecutado por el contenedor justo antes de quitar la aplicación.

En nuestro caso sólo sobrescribimos el primer método.

La implementación de la funcionalidad consiste primeramente en crear prototipos de los PE. Por ejemplo:

```
EvaluatorPE evaluatorPE = createPE(EvaluatorPE.class);
```

En el caso de que sólo se requiera una instancia del PE se pueden hacer dos cosas: enviarle eventos sin clave, o indicarlo de la siguiente manera:

```
metadataExtractorPE.setSingleton(true);
```

4.3.4.1. Envío de eventos entre elementos de procesado

Para crear un *stream* de eventos e indicar el PE de destino, usamos el método createStream(). Lo ilustramos con un ejemplo:

```
// Crea stream que tendra como destinatario ProbabilityCalculatorPE
Stream<DataEvent> probabilityQueryStream =
    createStream("ProbabilityQuery", new KeyFinder<DataEvent>() {
        @Override
        public List<String> get(final DataEvent arg0) {
            return ImmutableList.of(arg0.getId());
        }
    }, probabilityCalculatorPE);
```

Como podemos observar, el método recibe 3 parámetros:

- Nombre que identifica al *stream.*
- Objeto KeyFinder de tipo DataEvent con el método get() sobrescrito para devolver la clave de los eventos.
- PE al que van destinados los eventos.

En el ejemplo, cuando se inyecta un evento en el *stream* probabilityQueryStream, se identificará y recogerá su clave desde una instancia de la clase KeyFinder para luego buscarla entre las claves de las instancias del PE objetivo. En este caso, la clave del evento será lo que devuelve el método *getId()* de la clase DataEvent.

Para asignar un *stream* a un PE como *stream* de salida de los eventos emitidos, basta con invocar el método setDownstream() implementado en el PE con el objeto *stream* como parámetro. Por ejemplo:

```
metadataExtractorPE.setDownStream(probabilityQueryStream);
```

El *stream* de entrada a la aplicación proveniente del adaptador se crea de manera sencilla usando el método createInputStream(), que es una variación de createStream() para recibir eventos de otras aplicaciones; en este caso el adaptador. En este caso no hace falta indicar el parámetro KeyFinder, ya que los eventos son de clave nula y el PE destino solo tendrá una instancia. Cabe señalar que el nombre del *stream* ha de coincidir con el que se configure en el adaptador en tiempo de ejecución. Ejemplo:

```
createInputStream("Document", metadataExtractorPE);
```

También existe la posibilidad de enviar eventos a otras aplicaciones, de forma análoga a como se hace en el adaptador.

4.3.4.2. Uso de checkpoints para recuperación ante errores

Como vimos en el apartado 2.4.4.2, S4 proporciona un mecanismo de *checkpoints* para minimizar la pérdida de estado de un nodo que falle.

Si se quieren guardar *checkpoints* de un PE, podremos hacerlo de varias maneras. Por ejemplo existe la posibilidad de hacerlo cada cierto tiempo:

```
// Checkpoint de la instancia cada 30 segundos
metadataExtractorPE.setCheckpointingConfig(
    new CheckpointingConfig.Builder(CheckpointingMode.TIME)
    .frequency(30).timeUnit(TimeUnit.SECONDS).build());
```

También se puede hacer respaldo del estado del PE cada cierto número de eventos recibidos:

```
// Checkpoint de la instancia a cada evento recibido
evaluatorPE.setCheckpointingConfig(
    new CheckpointingConfig.Builder(CheckpointingMode.EVENT_COUNT)
    .frequency(1).build());
```

Para que un PE acepte *checkpoints* hay que tener en cuenta que su clase debe implementar un constructor sin argumentos. Además es importante saber que las propiedades *transient* no serán serializadas.

Existen acciones adicionales a realizar en tiempo de despliegue para activar los *checkpoints*. Estas se detallan en el próximo capítulo, donde se lleva a cabo todo lo relativo a la ejecución de la aplicación.

Capítulo 5.
Instalación y ejecución

En el capítulo anterior hemos visto lo esencial para llevar a cabo el desarrollo de una aplicación S4 de forma general, ejemplificando con código de nuestra propia implementación del desambiguador.

A continuación se expone todo lo necesario para poner a punto S4, desplegar la aplicación en las distintas máquinas del entorno, configurarla y finalmente ejecutarla.

5.1. CONFIGURACIÓN DEL ENTORNO

5.1.1. Ubicación compartida

Para el funcionamiento distribuido de la aplicación necesitamos un lugar de almacenamiento accesible desde todos los nodos, con el fin de ubicar ficheros comunes y los propios desplegables; así como para albergar los *checkpoints* de la aplicación de cara a poder recuperar el estado de los PE en caso de errores. En nuestro caso, para pruebas hemos utilizado un directorio compartido entre máquinas, que se ubicará en el directorio de trabajo del usuario con el nombre “sharedspace” (para ajustarnos a la ubicación especificada en el código fuente de nuestra aplicación).

Hay otras opciones para alojar estos contenidos:

- Los desplegables y ficheros comunes se pueden dejar en un servidor web accesible desde las máquinas.

- Los *checkpoints* se pueden almacenar en bases de datos, *datagrids* o memoria caché.
- Otra opción es usar un NAS si la escritura y lectura de datos es muy exigente, aunque hay que tener en cuenta que es más costoso.

Si tenemos la posibilidad de usar un directorio compartido entre máquinas nuestra implementación y la puesta a punto del entorno será mucho más sencilla. Es por ello que esta opción fue la elegida para nuestra aplicación.

Podemos montar el directorio compartido por ejemplo usando NFS [50] o SSHFS [51]. Ejemplo de montaje del directorio "sharedspace" en máquinas remotas usando SSHFS:

```
mkdir $HOME/sharedspace
sshfs <user>@<server>:<path>/sharedspace $HOME/sharedspace
```

Esta configuración tiene un inconveniente: si la máquina donde se encuentra físicamente el directorio tiene algún problema o se pierde la conexión con la misma, nuestra aplicación dejará de funcionar correctamente. Para un entorno de producción donde necesitemos alta disponibilidad usaremos otra estrategia para almacenar los datos que incluya algún mecanismo de replicación, como por ejemplo RAID [52] o HDFS [53].

5.1.2. Configuración de logs

S4 usa la librería Logback [54] para registrar *logs*. Lo que necesitamos saber de este módulo es lo que podemos modificar mediante configuración y cómo llevarlo a cabo.

La configuración por defecto se encuentra en el fichero <s4_home>/subprojects/s4-core/build/resources/main/logback.xml (salida estándar y nivel *debug*):

```
<configuration>
  <appender name="STDOUT"
class="ch.qos.logback.core.ConsoleAppender">
    <!-- encoders are assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by
default -->
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} -
%msg%n</pattern>
    </encoder>
  </appender>
  <root level="info">
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

- Nivel de logado → fijado en el atributo *level* de <*root*>. Los valores posibles son: *debug*, *info*, *warn*, *error*, *off*, *trace* y *all*.
- Salida → fijado en el atributo *class* del elemento <*appender*> cuyo atributo *name* está referenciado en <*root*>. Por defecto volcará los *logs* a la consola.
- Formato de las trazas → fijado en el <*pattern*> del <*appender*> en uso.

Para otras configuraciones, consultar el manual de Logback [55].

En teoría se puede modificar la configuración de *logs* a nivel aplicación, pero lo más sencillo es hacerlo a nivel global. Para ello seguiremos los siguientes pasos:

1. Editar el fichero <s4_home>/subprojects/s4-tools/build/install/s4-tools/bin/s4-tools. Añadir en la variable de entorno CLASSPATH la ruta a una carpeta específica donde meteremos nuestro fichero de configuración. Para aplicaciones distribuidas se recomienda usar un directorio compartido para no tener que editarlo en todas las máquinas.
2. Copiar el fichero de configuración por defecto al directorio referenciado anteriormente, cambiando su nombre a logback-test.xml.
3. Editar dicho fichero. La configuración de Logback cambiará desde este momento para todo aquello que se ejecute en S4.

5.1.3. Activación de checkpoints

Para activar el mecanismo de *checkpoints* con la implementación por defecto de S4, añadiremos el siguiente parámetro a la instrucción para desplegar la aplicación:

```
-emc=org.apache.s4.core.ft.FileSystemBackendCheckpointingModule
```

En un entorno distribuido, usando los *checkpoints* por defecto deberemos tener obligatoriamente un directorio compartido por todas las máquinas, ya sea por NFS, SSHFS o similar. Para indicarle al nodo dónde tiene que guardar/buscar los *checkpoints*, usaremos adicionalmente el siguiente parámetro con la ruta al directorio compartido:

```
-p s4.checkpointing.filesystem.storageRootPath=<shared_dir>
```

Existe la posibilidad de cambiar el *backend* de almacenamiento de los *checkpoints* (disco por defecto) a otros tipos: memoria caché, bases de datos, o *datagrids*. Para ello tendríamos que implementar la interfaz StateStorage, y añadir un nuevo módulo.

Por defecto S4 usa Kryo [56] para serializar y deserializar, aunque se pueden usar otros mecanismos sobrescribiendo los métodos checkpoint(), serializeState() y restoreState() de la clase ProcessingElement.

Los PE permiten *checkpoints* cuando se activa su *flag* "sucio", algo que se produce al procesar un evento. Este *flag* se puede comprobar mediante isDirty(), y borrar con clearDirty() cuando queramos. En algunos casos, dependiendo de la aplicación, sólo algunos de los eventos recibidos por un PE cambian su estado interno. Deberemos sobrescribir los métodos anteriores para evitar *checkpoints* innecesarios.

El *framework* de *checkpoints* tiene varios parámetros que podemos sobrescribir para cambiar sus valores por defecto. Para ello se recomienda consultar el apéndice B.2.

5.1.4. Inclusión de librerías

S4 nos da la posibilidad de incluir librerías adicionales en nuestra aplicación de forma sencilla.

Para añadir librerías externas que están en repositorios Maven (las más extendidas), simplemente tenemos que añadir el ID del artefacto en el fichero build.gradle de nuestra aplicación.

Por ejemplo, para añadir a nuestro proyecto las librerías Gson (anteriormente nombrada) y Jedis [57] (librería que implementa un cliente de Redis), añadiremos las siguientes líneas dentro del array project.ext["libraries"]:

```
/* All project libraries must be defined here. */
project.ext["libraries"] = [
   s4_base:            'org.apache.s4:s4-base:'+ s4Version,
   s4_comm:            'org.apache.s4:s4-comm:'+ s4Version,
   s4_core:            'org.apache.s4:s4-core:'+ s4Version,
   Gson:               'com.google.code.gson:gson:2.2.2',
   jedis:              'redis.clients:jedis:2.1.0'
]
```

Una buena fuente para encontrar el ID del artefacto es acudir al repositorio central de Maven [58].

También escribiremos lo siguiente en ese fichero para indicar que la dependencia se introduce en tiempo de compilación:

```
dependencies {
   compile (libraries.s4_base)
   compile (libraries.s4_comm)
   compile (libraries.s4_core)
   compile (libraries.Gson)
   compile (libraries.jedis)
}
```

Además, si usamos Eclipse para desarrollo, tendremos que actualizar el *classpath* del proyecto ejecutando:

```
./gradlew eclipse
```

En caso de usar IntellijIDEA:

```
./gradlew idea
```

Las dependencias de la aplicación serán incluidas automáticamente en el fichero .s4r que se despliega en los nodos.

Para añadir librerías externas no publicadas en repositorios Maven, tendremos que hacer cualquiera de estas dos cosas:

a) publicarlas en nuestro repositorio Maven local
b) simplemente añadirlas al directorio “lib”

Igualmente tendremos que modificar el fichero build.gradle de nuestra aplicación para meterlas como dependencias en tiempo de compilación.

5.1.5. Consulta de estado y obtención de métricas

S4 proporciona la opción de revisar en todo momento el estado de los clústeres de nuestra plataforma. Para ello podemos hacer uso del comando *status*, que nos dará información relativa a los clústeres, sus nodos activos, y los *streams* existentes entre aplicaciones.

Ejemplo:

```
App Status
--------------------------------------------------------------------------------------------
        Name              Cluster                              URI
--------------------------------------------------------------------------------------------
 disambiguator-adapter    cluster2      file:/root/sharedspace/disambiguator-adapter.s4r
   disambiguator          cluster1      file:/root/sharedspace/disambiguator.s4r
--------------------------------------------------------------------------------------------
```

```
Cluster Status
-------------------------------------------------------------------------------------------------
                                                          Active nodes
    Name                App           Tasks    ---------------------------------------------
                                                 Number    Task id          Host        Port
-------------------------------------------------------------------------------------------------
  cluster2     disambiguator-adapter   1          1        Task-0      ubuntu-server    13000
  cluster1         disambiguator       1          1        Task-0      ubuntu-server    12000
-------------------------------------------------------------------------------------------------


Stream Status
-------------------------------------------------------------------------------------------------
        Name                    Producers                                Consumers
-------------------------------------------------------------------------------------------------
      Document       cluster2(disambiguator-adapter)             cluster1(disambiguator)
-------------------------------------------------------------------------------------------------
```

S4 además monitoriza la parte interna del sistema y recopila estadísticas continuamente en tiempo de ejecución, con el fin de conseguir procesar grandes cantidades de eventos con bajo retardo. Para llevar esto a cabo, usa la librería Metrics [59] de Java; elegida sobre otros métodos por su eficiencia.

Por defecto en S4 se monitorizan cachés, *checkpoints*, recepción y emisión de eventos, y colas. Adicionalmente se pueden monitorizar los PE añadiendo sondas de forma sencilla; S4 reportará nuestras propias estadísticas junto a todas las demás (las métricas son expuestas por cada nodo mediante JMX [60]).

El parámetro de configuración s4.metrics.config activa el volcado periódico de las estadísticas a la consola o a un fichero con formato CSV [61]. Este parámetro se especifica como un parámetro de la aplicación (opción del comando *deploy*) o del nodo (opción del comando *node*), y debe ajustarse a la siguiente expresión regular:

```
(csv:.+|console):(\d+):(DAYS|HOURS|MICROSECONDS|MILLISECONDS|MINUTE
S|NANOSECONDS|SECONDS)
```

Un ejemplo volcando a la consola:

```
console:1:MINUTES
```

Otro ejemplo volcando a fichero:

```
csv:/ruta/a/directorio:30:SECONDS
```

Para el caso del volcado a fichero se recomienda hacerlo a nivel de nodo, y especificando un directorio distinto para cada uno (y vacío).

5.1.6. Consejos adicionales

Como último apunte con respecto al entorno, es recomendable tomar precauciones a la hora de ejecutar nuestras aplicaciones:

- Revisar los cortafuegos de las máquinas donde se ejecute la aplicación. El tráfico a determinados puertos podría estar restringido.
- Asegurarnos de que S4 es capaz de resolver la IP local, en lugar de 127.0.0.1. Si existe algún problema con esto, tendremos que tocar el fichero /etc/hosts.
- Usar el comando *status* de S4 ante cualquier fallo de una aplicación, para comprobar el despliegue

En el siguiente punto se detalla la configuración del desambiguador, así como los pasos necesarios para desplegar y ejecutar la aplicación en un entorno distribuido.

5.2. INSTALACIÓN Y EJECUCIÓN DE LA APLICACIÓN

5.2.1. Consideraciones previas

El código de los dos módulos implementados (el de la aplicación en sí, "disambiguator", y el del adaptador, "disambiguator-adapter") podría situarse en cualquier carpeta, pero se recomienda ubicarlo en $HOME/s4-0.6.0/test-apps/ (como se indicó en la creación del esqueleto) para facilitar el proceso con las instrucciones que se detallan en este apartado.

Si tenemos dependencias de código entre el módulo de la aplicación y el del adaptador (como es el caso con el paquete es.uc3m.it.jduran.disambiguator.structures), en la máquina donde tengamos los ficheros fuente y se creen los paquetes .s4r, duplicaremos el directorio en cuestión o crearemos un enlace simbólico para que desde un módulo se vean clases del otro (requisito necesario para la compilación):

```
ln -s $HOME/s4-0.6.0/test-
apps/disambiguator/src/main/java/es/uc3m/it/jduran/disambiguator/st
ructures $HOME/s4-0.6.0/test-apps/disambiguator-
adapter/src/main/java/es/uc3m/it/jduran/disambiguator/structures
```

Dejaremos también en el almacenamiento compartido los siguientes ficheros necesarios para la ejecución de la aplicación:

- corpus.tsv: datos de entrada siguiendo la especificación explicada en el apartado 3.1.
- disambiguator.config: fichero de configuración siguiendo la especificación detallada en el apartado 5.2.2.

Por último, necesitamos un servidor de Redis funcionando en la máquina configurada para albergarlo (lo instalaremos si es necesario siguiendo las instrucciones del apéndice C.3). Este servidor es necesario para que el adaptador de la entrada y la aplicación en sí compartan información de cara a evaluar la eficiencia del desambiguador.

Para arrancarlo, usamos el siguiente comando:

```
$HOME/redis-2.6.13/src/redis-server
```

Si se requiere alta disponibilidad, podría usarse un clúster de Redis o utilizar otra estrategia similar con replicación.

Es posible que la aplicación, al procesar el corpus entero, consuma más memoria RAM de la disponible. Para ello se proponen 2 soluciones:

- Aumentar la memoria de la máquina virtual de Java si se tiene esa posibilidad.
- Usar más máquinas para desplegar la aplicación, y aprovechar así todo el potencial de S4. En el caso de nuestro corpus de entrada, ejecutando la aplicación principal en una sola máquina se llegó a alcanzar el límite de memoria RAM de la máquina virtual de Java, situado en 2GB. En ese punto el nodo se colapsa y Zookeeper acaba perdiendo contacto con él. La solución fue simplemente usar una máquina adicional para la aplicación y no se tuvieron más problemas de este tipo.

Si ejecutamos el servidor de Zookeeper que viene incluido en S4, no se garantiza alta disponibilidad. Para dotar a nuestra aplicación con esta característica es necesario instalar Zookeeper de forma independiente en todas las máquinas siguiendo las instrucciones del anexo C.4.

5.2.2. Configuración

Existen varias cosas que podemos configurar en nuestra aplicación por medio del fichero **disambiguator.config** dentro del directorio compartido:

- redis-server → nombre o IP de la máquina donde se aloja el servidor Redis.
- corpus-file → nombre del fichero con el corpus de entrada (en la ruta del directorio compartido)
- ambiguity-keys → entidades del corpus que se sustituirán por un seudónimo ambiguo.

Ejemplo:

```
redis-server=news.gast.it.uc3m.es
corpus-file=corpus.tsv
ambiguity-keys=los_angeles_lakers.org,chicago_bulls.org
```

Este fichero no se debe editar en caliente.

5.2.3. Ejecución

Estos son los pasos necesarios para ejecutar la aplicación desde cero.

1. Situarnos en el directorio base de S4:

```
cd $HOME/s4-0.6.0/
```

2. Arrancar una instancia de Zookeeper mediante el comando *zkServer*. El parámetro *clean* sirve para hacerlo de manera limpia, eliminando posibles restos de una ejecución anterior:

```
./s4 zkServer -clean &

INFO  ZKServer: Starting zookeeper server on port [2181]
INFO  ZKServer: cleaning existing data in [/tmp/tmp/zookeeper/data]
        and [/tmp/tmp/zookeeper/log]
INFO  ZKServer: Zookeeper server started on port [2181]
```

   Si queremos alta disponibilidad, en lugar de esto, ejecutaremos lo siguiente en todos nuestros servidores dentro del directorio donde se haya instalado Zookeeper:

```
bin/zkServer.sh start
```

3. Crear 2 clústeres; el principal donde residirá la aplicación en sí ("cluster1"), y otro adicional para el adaptador ("cluster2"). Para ello usaremos el comando *newCluster*. El parámetro *nbTasks* fijará el número de particiones, o lo que es lo

mismo, el número máximo de nodos activos en cada clúster. El parámetro *flp* indica el puerto inicial a partir del cual escucharán los distintos nodos del clúster.

Si usamos por ejemplo 1 nodo para el adaptador y 2 para la aplicación:

```
./s4 newCluster -c=cluster1 -nbTasks=2 -flp=12000

INFO  DefineCluster: preparing new cluster [cluster1] with [2]
node(s)
INFO  DefineCluster: New cluster configuration uploaded into
zookeeper
```

```
./s4 newCluster -c=cluster2 -nbTasks=1 -flp=13000

INFO  DefineCluster: preparing new cluster [cluster2] with [1]
node(s)
INFO  DefineCluster: New cluster configuration uploaded into
zookeeper
```

4. Arrancar tantos nodos como se deseen en el clúster principal por medio del comando *node*. El parámetro *c* sirve para indicar el nombre del clúster donde se arranca el nodo. El parámetro *zk* especifica la cadena de conexión a Zookeeper (por defecto *localhost:2181*).

   Si Zookeeper se está ejecutando en la misma máquina donde se va a lanzar el nodo, ejecutaremos lo siguiente:

```
./s4 node -c=cluster1

INFO  AssignmentFromZK: New session:91010794798645251; state is :
       SyncConnected
INFO  AssignmentFromZK: Successfully acquired task:Task-0 by
ubuntu-server
```

   Si por el contrario Zookeeper se encuentra en una máquina remota, ejecutaremos esto otro:

```
./s4 node -c=cluster1 -zk=<host_zookeeper>:2181
```

Resultado después de crear 2 nodos:

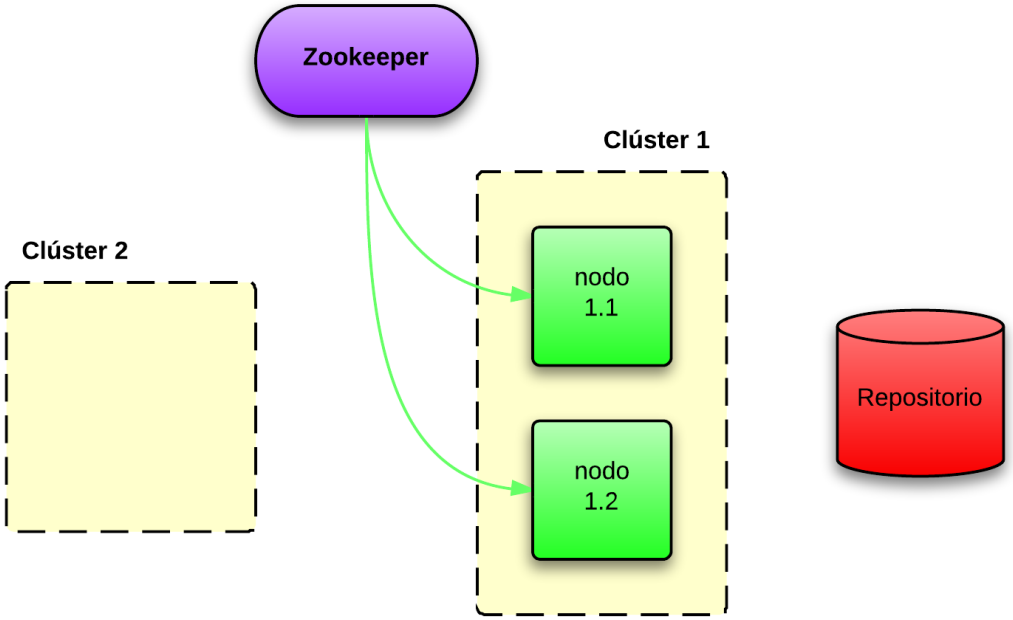

Ilustración 16. Despliegue (1/5)

5. Arrancar nodo para el adaptador, indicando el *stream* de salida con sólo sobrescribir la propiedad *s4.adapter.output.stream* mediante el parámetro *p*:

```
./s4 node -c=cluster2 -p=s4.adapter.output.stream=Document

INFO  ParsingUtils$InlineConfigParameterConverter: processing
inline
        configuration parameter s4.adapter.output.stream=Document
INFO  AssignmentFromZK: New session:91010794798645252; state is :
        SyncConnected
INFO  AssignmentFromZK: Successfully acquired task:Task-0 by
ubuntu-server
```

Resultado después de crear 1 nodo:

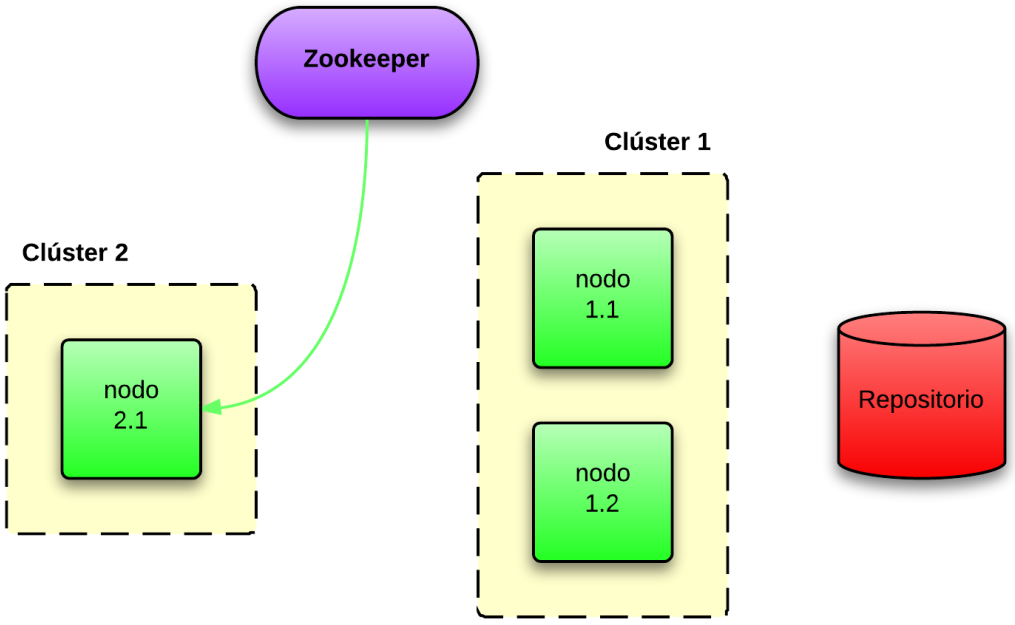

Ilustración 17. Despliegue (2/5)

6. Crear los ficheros a desplegar en la máquina donde tengamos los fuentes de la aplicación (es recomendable que sea la misma donde se ejecuta Zookeeper) por medio del comando *s4r*. El parámetro *b* sirve para dar la ruta del fichero build.gradle, necesario para construir la aplicación. El parámetro *a* sirve para indicar el nombre completo de la clase principal:

```
./s4 s4r -b=`pwd`/test-apps/disambiguator/build.gradle
disambiguator -a=es.uc3m.it.jduran.disambiguator.DisambiguatorApp

./s4 s4r -b=`pwd`/test-apps/disambiguator-adapter/build.gradle
disambiguator-adapter -
a=es.uc3m.it.jduran.disambiguator.ArticlesInputAdapter
```

Sólo nos queda mover los ficheros a la ubicación elegida (ya sea un directorio compartido como es nuestro caso, o un servidor web accesible desde todas las máquinas del entorno).

```
mv test-apps/disambiguator/build/libs/disambiguator.s4r
$HOME/sharedspace/

mv test-apps/disambiguator-adapter/build/libs/disambiguator-
adapter.s4r $HOME/sharedspace/
```

7. Desplegar la aplicación disambiguator en el clúster principal (se desplegará automáticamente en todos sus nodos activos a pesar de estar en distintas máquinas). Para ello usamos el comando *deploy*. El parámetro *s4r* sirve para

dar la ruta del desplegable, el parámetro *c* para indicar el clúster donde se ha de realizar el despliegue, y el parámetro *appName* para definir el nombre de la aplicación. Si queremos activar los checkpoints, tendremos que seguir las instrucciones del apartado 5.1.3.

```
./s4 deploy -s4r=$HOME/sharedspace/disambiguator.s4r -c=cluster1 -
appName=disambiguator

INFO  Deploy: Using specified S4R [file:../sharedspace/disambiguator.s4r]
```

Esto hace que se desencadenen una serie de procesos en los nodos del clúster principal:

```
INFO  S4Bootstrap: Initializing S4 app with : []]
INFO  S4Bootstrap: Loading application [disambiguator] from file
        [/tmp/tmp8744325187905052155s4r]
INFO  S4RLoaderFactory: Unzipping S4R archive in
[/tmp/1388715354824-
        0/tmp8744325187905052155s4r-1388715354826]
INFO  Bootstrap: App class name is:
        es.uc3m.it.jduran.disambiguator.DisambiguatorApp
INFO  ClusterFromZK: Changing cluster topology to {

nbNodes=1,name=cluster1,mode=unicast,type=,nodes=[{partition=0,
        port=12000,machineName=ubuntu-server,taskId=Task-0}]} from
null
(...)
INFO  ClustersFromZK: Detected new stream [Document]
INFO  App: Init prototype
[es.uc3m.it.jduran.disambiguator.EvaluatorPE]
INFO  App: Init prototype
[es.uc3m.it.jduran.disambiguator.DisambiguatorPE]
INFO  App: Init prototype
        [es.uc3m.it.jduran.disambiguator.ProbabilityCalculatorPE]
INFO  App: Init prototype
        [es.uc3m.it.jduran.disambiguator.MetadataExtractorPE]
```

Resultado:

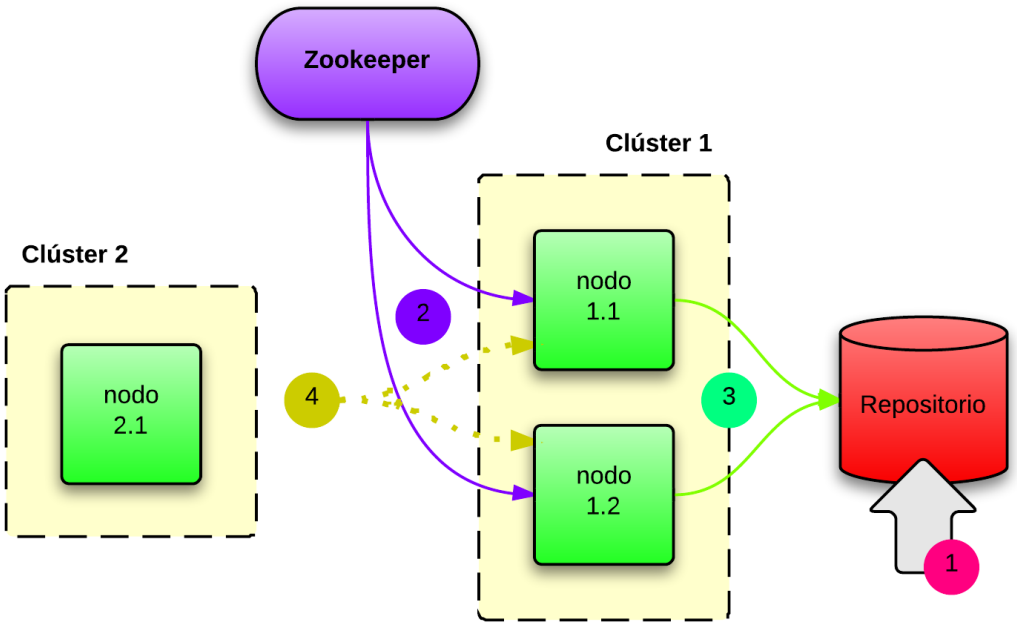

Ilustración 18. Despliegue (3/5)

1) La aplicación S4 se ubica en el repositorio (en nuestro caso el sistema de ficheros local).
2) Se avisa a los nodos del clúster principal de que existe una nueva aplicación.
3) Los nodos obtienen, cargan y arrancan la aplicación.
4) Se crea un nuevo *stream* con un determinado nombre y ambos nodos se registran como consumidores del mismo.

8. Desplegar la aplicación disambiguator-adapter en el clúster 2:

```
./s4 deploy -s4r=$HOME/sharedspace/disambiguator-adapter.s4r -
c=cluster2 -appName=disambiguator-adapter

INFO  Deploy: Using specified S4R [file:/root/sharedspace/disambiguator-
       adapter.s4r]
```

Resultado:

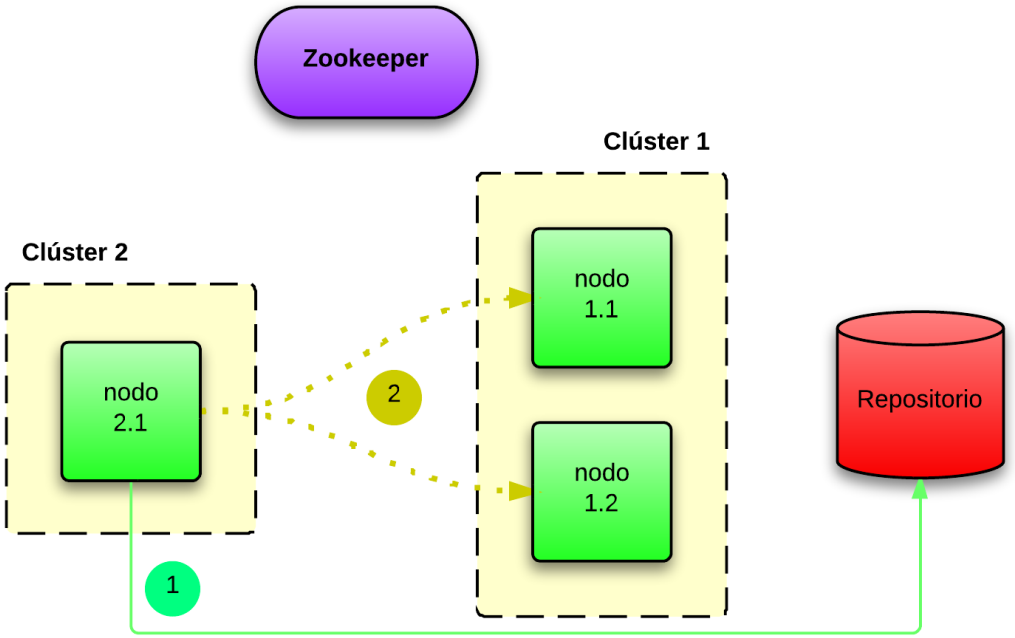

Ilustración 19. Despliegue (4/5)

1) El nodo en el segundo clúster carga la aplicación del adaptador.
2) Esta aplicación identifica a los nodos consumidores de su *stream* de salida.

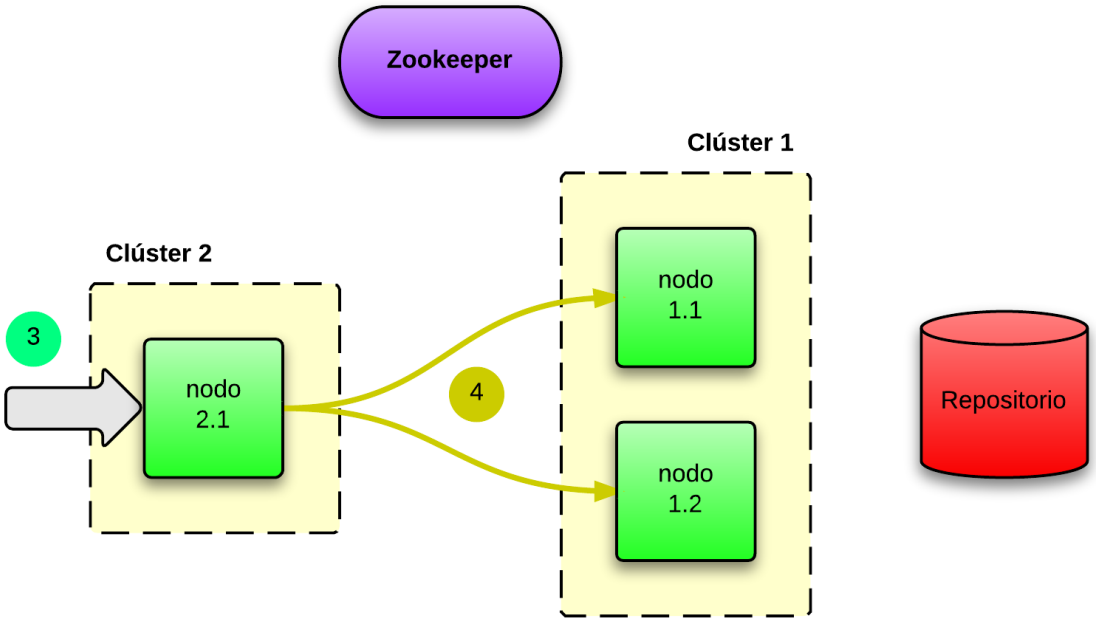

Ilustración 20. Despliegue (5/5)

3) Los datos de entrada empiezan a ser leídos por el adaptador, que los procesa y crea eventos S4.

4) Los eventos son distribuidos a uno de los nodos consumidores; aquél donde se instancie el PE de entrada con la llegada del primer evento.

9. Observamos nuestros resultados en el nodo que acoge a la instancia de EvaluatorPE:

```
...
12:52:48.286 INFO Entidad desambiguada: los_angeles_-calif-.loc -
Entidad correcta: los_angeles_-calif-.loc - ACIERTO!
12:52:48.286 INFO  Estimaciones del desambiguador: 18381
12:52:48.286 INFO  Aciertos del desambiguador: 16736
12:52:48.286 INFO  Efectividad del desambiguador: 91.05%
...
```

En el siguiente capítulo se lleva a cabo una validación completa de la aplicación y sus resultados.

Capítulo 6.
Validación

De cara a probar, analizar y valorar nuestra aplicación final, se implementaron varios casos de prueba y ejemplos prácticos extraídos de un primer análisis sobre el contenido del corpus original.

Los casos de prueba fueron usados para depurar y probar determinadas situaciones con datos de entrada controlados, de forma que quedase validada la implementación.

Los ejemplos de ejecución nos ayudan a extraer resultados y sacar conclusiones de la eficiencia de nuestra aplicación con casos reales.

Como punto de partida de cada prueba o ejemplo, basta con fijar las dos entidades que el adaptador de la entrada cambiará por un seudónimo ambiguo.

Para evaluar la eficiencia de la aplicación, se ha intentado recoger un buen abanico de pares para desambiguación, usando para ello entidades que aparecen bastante a lo largo de todo el corpus. Las desambiguaciones elegidas cubren los siguientes casos:

- Pares de entidades con muchas apariciones, de alguna manera relacionadas por la temática (ambas son países, ciudades, personajes, etc).
- Pares de entidades no coincidentes en el tiempo, lo que provoca de forma generalizada que durante un primer intervalo sólo aparezca una, y en un segundo intervalo la predominante sea la otra (con pequeñas salvedades). Además se han elegido entidades que comparten el mismo contexto (por ejemplo presidentes de Estados Unidos), de forma que cuando empiece a aparecer en los artículos la segunda entidad, la desambiguación será realmente

complicada. Es probablemente el caso más difícil estudiado para la aplicación en esta memoria.

- Pares de entidades con un muy diferente grado de aparición.
- Pares de entidades con pocas apariciones para ambas.

Como último paso en la evaluación de la plataforma se han llevado a cabo pruebas para validar el mecanismo de respuesta ante errores; abarcando desde la detección de un nodo caído hasta la recuperación del estado mediante *checkpoints*, pasando por el proceso de activación de un nodo de respaldo.

6.1. CASOS DE PRUEBA

Estos casos de prueba se ejecutaron con una máquina para el adaptador y otra para la aplicación en sí. Cada uno de los tests cuenta con un corpus de entrada personalizado para la ocasión, con las entradas reflejadas a continuación. Las entidades configuradas para crear el seudónimo ambiguo en el adaptador son “Madrid” y “Barcelona”. Se incluyen los resultados obtenidos (que en todos los casos expuestos coinciden con los resultados esperados).

6.1.1. Descarte de entradas con dos entidades ambiguas

Cuando aparecen las dos entidades configuradas para crear el seudónimo ambiguo en un mismo artículo, este se debe descartar.

Entradas

Artículo	Entidades
1	Madrid, Barcelona

Resultados

Artículo	Estimación de probabilidades	Entidad desambiguada	Resultado
1	-	-	Descartado ✓

6.1.2. Evolución de las decisiones. Influencia de apariciones individuales

El número de apariciones de una de las dos entidades configuradas será el factor determinante cuando el resto de entidades del contexto no se repiten a lo largo de los distintos artículos.

Entradas

Artículo	Entidades
1	Madrid, Móstoles
2	Madrid, Leganés
3	Barcelona, Hospitalet
4	Barcelona, Sitges
5	Barcelona, Cornellá
6	Madrid, Getafe

Resultados

Artículo	Estimación de probabilidades	Entidad desambiguada	Resultado	
1	Pr(Madrid) = Pr(Barcelona)	?	?	✓
2	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
3	Pr(Madrid) > Pr(Barcelona)	Madrid	Fallo	✓
4	Pr(Madrid) > Pr(Barcelona)	Madrid	Fallo	✓
5	Pr(Madrid) = Pr(Barcelona)	?	?	✓
6	Pr(Madrid) < Pr(Barcelona)	Barcelona	Fallo	✓

6.1.3. Evolución de las decisiones. Incidencia de apariciones conjuntas

El número de apariciones de las entidades del contexto son el factor determinante cuando las entidades que dan lugar a la ambigüedad han aparecido el mismo número de veces hasta el momento de la desambiguación.

Entradas

Artículo	Entidades
1	Madrid, Móstoles
2	Madrid, Leganés
3	Barcelona, Hospitalet
4	Barcelona, Sitges
5	Madrid, Hospitalet

Resultados

Artículo	Estimación de probabilidades	Entidad desambiguada	Resultado	
1	Pr(Madrid) = Pr(Barcelona)	?	?	✓
2	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
3	Pr(Madrid) > Pr(Barcelona)	Madrid	Fallo	✓
4	Pr(Madrid) > Pr(Barcelona)	Madrid	Fallo	✓
5	Pr(Madrid) < Pr(Barcelona)	Barcelona	Fallo	✓

6.1.4. Evolución de las decisiones. Prevalencia del contexto sobre apariciones individuales

La reaparición de una entidad del contexto asociada a una de las dos entidades que forman la ambigüedad tiene más peso en la decisión que una aparición más de la otra entidad que forma la ambigüedad.

Entradas

Artículo	Entidades
1	Madrid, Móstoles
2	Madrid, Leganés
3	Barcelona, Hospitalet
4	Barcelona, Sitges
5	Barcelona, Cornellá
6	Madrid, Móstoles

Resultados

Artículo	Estimación de probabilidades	Entidad desambiguada	Resultado	
1	Pr(Madrid) = Pr(Barcelona)	?	?	✓
2	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
3	Pr(Madrid) > Pr(Barcelona)	Madrid	Fallo	✓
4	Pr(Madrid) > Pr(Barcelona)	Madrid	Fallo	✓
5	Pr(Madrid) = Pr(Barcelona)	?	?	✓
6	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓

6.1.5. Evolución de las decisiones. Más situaciones

Una mezcla de casos anteriores.

Entradas

Artículo	Entidades
1	Valencia, Spain
2	Bilbao, Spain
3	Madrid, Valencia
4	Barcelona, Bilbao
5	Madrid, Valencia, Bilbao
6	Barcelona, Bilbao

Resultados

Artículo	Estimación de probabilidades	Entidad desambiguada	Resultado	
1	(sólo entrenamiento)	-	-	-
2	(sólo entrenamiento)	-	-	-
3	Pr(Madrid) = Pr(Barcelona)	?	?	✓
4	Pr(Madrid) > Pr(Barcelona)	Madrid	Fallo	✓
5	Pr(Madrid) = Pr(Barcelona)	?	?	✓
6	Pr(Madrid) > Pr(Barcelona)	Madrid	Fallo	✓

6.1.6. Evolución de la efectividad

La efectividad final del desambiguador se comporta de forma coherente cuando la entidad que crea la ambigüedad es siempre la misma y el contexto no influye porque nunca se repite.

Entradas

Artículo	Entidades
1	Madrid, Entity1, Entity01
2	Madrid, Entity2, Entity02
3	Madrid, Entity3, Entity03
4	Madrid, Entity4, Entity04
5	Madrid, Entity5, Entity05
6	Madrid, Entity6, Entity06
7	Madrid, Entity7, Entity07
8	Madrid, Entity8, Entity08
9	Madrid, Entity9, Entity09
10	Madrid, Entity0, Entity00

Resultados

Artículo	Estimación de probabilidades	Entidad desambiguada	Resultado	
1	Pr(Madrid) = Pr(Barcelona)	?	?	✓
2	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
3	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
4	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
5	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
6	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
7	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
8	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
9	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓
10	Pr(Madrid) > Pr(Barcelona)	Madrid	Acierto	✓

Efectividad = 90%-100% (dependiendo de si para la primera desambiguación tenemos fallo o acierto).

Con estos tests queda probada la funcionalidad básica de la aplicación. A continuación se evalúa el algoritmo implementado con ejemplos prácticos usando el corpus de entrada.

6.2. EJEMPLOS PRÁCTICOS Y RESULTADOS

Para cada uno de los casos prácticos analizados en este punto, se muestra la siguiente información:

- **Entidades ambiguadas**: las 2 entidades elegidas para crear el seudónimo ambiguo a partir de los datos de entrada, así como el número de artículos del corpus donde aparece cada una de ellas.
- **Resultados**: una tabla con el número total de estimaciones realizadas (igual al número de artículos del corpus donde aparecía una y sólo una de las dos entidades elegidas), la cantidad exacta de aciertos y la efectividad del desambiguador para una ejecución completa del proceso usando el corpus de entrada.
  Con el fin de poder valorar la efectividad del algoritmo Naïve Bayes en términos relativos, se indica también la efectividad obtenida fruto de ejecutar el mismo proceso de desambiguación usando un algoritmo sencillo, que simplemente resuelve cada ambigüedad escogiendo la entidad más frecuente durante el entrenamiento previo. Esta efectividad aparece en las tablas como “Efectividad MF”.
  No existe la necesidad de reflejar el resultado de varias ejecuciones debido a que el algoritmo es prácticamente determinista; la única componente de azar viene dada por las decisiones que toma el desambiguador en caso de empate entre las probabilidades de ambas entidades; algo que ocurrirá por ejemplo en la desambiguación del primer artículo. El número de decisiones al azar queda reflejado en los resultados para tener constancia del margen de error en la efectividad.
- **Comentarios**: notas adicionales sobre los resultados.

A continuación se exponen todos los ejemplos elegidos.

6.2.1. Ejemplo práctico 1

- Entidades ambiguadas:
  - **new_york_state.loc** (27.802 artículos)
  - **new_jersey.loc** (22.970 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
47.580	2	42.680	**89,70%**	55,08%

- Comentarios:
  Estas dos entidades contaban con muchas apariciones en el corpus de entrada, así que se esperaba obtener una buena efectividad. Y así fue; el desambiguador en este caso roza el 90% de aciertos. Comparando con la efectividad que se obtendría eligiendo siempre la entidad más frecuente, estamos consiguiendo una gran mejora con el uso de Naïve Bayes.
  El margen de error en la cifra de la efectividad viene dado por el número de decisiones al azar del desambiguador; en este caso un despreciable 0,004%.

6.2.2. Ejemplo práctico 2

- Entidades ambiguadas:
  - **bush_george_-pres-.per** (10.184 artículos)
  - **bush_george_w_-pres-.per** (22.154 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
32.338	1	29.385	**90,87%**	68,51%

- Comentarios:
  Estamos ante un caso de ambigüedad complejo, ya que ambas entidades (George Bush presidente, y George W. Bush presidente) no son contemporáneas, y sus intervalos de aparición están muy definidos. Además este es probablemente uno de los casos más complicados para el algoritmo implementado por nuestra aplicación, al menos a priori, ya que el contexto para ambas entidades es muy parecido.
  No obstante, la efectividad conseguida es muy elevada a pesar de contar con dichas dificultades, y mejora notablemente al algoritmo donde no se computa el contexto, por lo que nuestro algoritmo sale reforzado de esta prueba.

6.2.3. Ejemplo práctico 3

- Entidades ambiguadas:
  - **clinton_bill_-pres-.per** (23.483 artículos)
  - **bush_george_w_-pres-.per** (22.154 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
45.449	1	38.553	**84,83%**	51,46%

- Comentarios:
  De nuevo un caso con dos entidades muy similares e intervalos de aparición muy definidos. Este caso es si cabe más complicado que el anterior, ya que los intervalos de tiempo son consecutivos (George W. Washington sucedió en el cargo de presidente a Bill Clinton).
  La efectividad de la aplicación aquí es muy alta pero un poco peor, debido principalmente a que en el caso anterior había un intervalo de 8 años entre los gobiernos de George Bush padre y George Bush hijo, mientras que aquí no existe dicha separación, por lo que el contexto es aún más parecido para estas entidades, siendo más complicado discernir entre ambas.

6.2.4. Ejemplo práctico 4

- Entidades ambiguadas:
  - **iraq.loc** (24.892 artículos)
  - **united_states.loc** (58.457 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
81.788	1	74.845	**91,51%**	70,52%

- Comentarios:
  En este caso, al igual que en los siguientes, tenemos dos entidades que aparecen bastantes veces juntas, por lo que también pueden tener un contexto similar en determinados casos. Aun así el resultado es muy bueno y supera con creces lo obtenido para el algoritmo donde se escoge siempre la entidad más frecuente.

6.2.5. Ejemplo práctico 5

- Entidades ambiguadas:
  - **europe.loc** (10.706 artículos)
  - **china.loc** (11.377 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
21.379	1	18.476	**86,42%**	51,73%

6.2.6. Ejemplo práctico 6

- Entidades ambiguadas:
  - **great_britain.loc** (13.454 artículos)
  - **japan.loc** (14.001 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
26.632	1	23.688	**88,95%**	50,95%

6.2.7. Ejemplo práctico 7

- Entidades ambiguadas:
  - **france.loc** (9.346 artículos)
  - **germany.loc** (10.252 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
14.032	1	11.752	**83,75%**	59,31%

6.2.8. Ejemplo práctico 8

- Entidades ambiguadas:
  - **iran.loc** (7.167 artículos)
  - **afghanistan.loc** (6.334 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
13.151	1	11.975	**91,06%**	53,18%

6.2.9. Ejemplo práctico 9

- Entidades ambiguadas:
  - **washington_-dc-.loc** (13.957 artículos)
  - **los_angeles_-calif-.loc** (4.569 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
18.381	1	16.765	**91,21%**	75,47%

6.2.10. Ejemplo práctico 10

- Entidades ambiguadas:
  - **bin_laden_osama.per** (1.426 artículos)
  - **hussein_saddam_-pres-.per** (3.797 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
5.127	1	4.924	**96,04%**	73,10%

- Comentarios:
  En este caso el seudónimo se crea a partir de dos entidades que podrían compartir contexto en determinados casos, tratándose de un medio de Estados Unidos. Los resultados de la aplicación son sin embargo excelentes.

6.2.11. Ejemplo práctico 11

- Entidades ambiguadas:
  - **yeltsin_boris_n_-pres-.per** (1.907 artículos)
  - **pope_-john_paul_ii-.per** (1.419 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
3.322	1	3.261	**98,16%**	57,31%

- Comentarios:
  El contexto de ambas entidades es bastante diferente; es por ello que la efectividad es muy elevada.

6.2.12. Ejemplo práctico 12

- Entidades ambiguadas:
  - **los_angeles_lakers.org** (1.177 artículos)
  - **chicago_bulls.org** (1.511 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
2.433	4	2.025	**83,23%**	54,38%

- Comentarios:
  Efectividad alta, aunque sensiblemente menor que en casos anteriores. Esto se debe a que el contexto de ambas entidades es muy parecido, ya que corresponden a dos equipos de baloncesto que juegan en la misma liga.

6.2.13. Ejemplo práctico 13

- Entidades ambiguadas:
  - **new_york_city.loc** (132.475 artículos)
  - **barcelona_-spain-.loc** (264 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
132.732	1	132.413	**99,76%**	99,80%

- Comentarios:
  En este caso las entidades elegidas para crear la ambigüedad cuentan con mucha diferencia de apariciones en el corpus de entrada. Por ello la efectividad es muy elevada.

Podemos observar otra consecuencia fruto del desequilibrio entre entidades; y es que la efectividad del desambiguador en este caso es casi idéntica a la obtenida con el algoritmo donde se elige siempre la entidad más frecuente. Incluso podemos ver que la efectividad es ligeramente inferior, debido a una particularidad del suavizado de Naïve Bayes que implica malas desambiguaciones cuando al desequilibrio anterior se le suma la llegada de artículos con un contexto muy extenso y formado al completo por entidades no aparecidas en entrenamiento.

6.2.14. Ejemplo práctico 14

- Entidades ambiguadas:
  - **madrid_-spain-.loc** (403 artículos)
  - **barcelona_-spain-.loc** (264 artículos)

- Resultados:

Estimaciones	Al azar	Aciertos	Efectividad	(Efectividad MF)
647	5	525	**81,14%**	60,12%

- Comentarios:
  Estamos ante dos entidades con pocas apariciones en el corpus de entrada. La efectividad es más baja que en casos anteriores, debido de nuevo a que los contextos son muy similares tratándose de un medio estadounidense. Además el número de muestras de entrenamiento es más reducido.

Como podemos ver analizando los resultados obtenidos en estas pruebas, la efectividad conseguida por nuestra aplicación de desambiguación es bastante elevada incluso en los casos más complejos. Antes de sacar las conclusiones finales sobre su rendimiento, también es conveniente analizar la fiabilidad y comportamiento ante errores que pudieran surgir en la plataforma; algo que se lleva a cabo en el siguiente apartado.

6.3. PRUEBAS DE RESPUESTA ANTE ERRORES

Como ya hemos visto, S4 proporciona un mecanismo automático de recuperación ante fallos en el sistema. No es necesario activar nada para que las aplicaciones desarrolladas sobre la plataforma se beneficien de dicho mecanismo.

Además S4 nos da la posibilidad de implementar en nuestra aplicación el respaldo de los PE mediante *checkpoints*, lo que permite minimizar la pérdida de datos, ya que en caso de fallo se recupera una copia reciente del estado de los PE en el nodo caído.

En este apartado se pone a prueba el funcionamiento de estos mecanismos usando nuestra aplicación.

6.3.1. Detección de nodo caído

Teniendo la aplicación del desambiguador ejecutándose sobre 2 nodos activos en el clúster principal, identificaremos como "nodo 2" al que contiene el elemento de procesado final (EvaluatorPE), y como "nodo 1" al otro.

Los elementos de procesado DisambiguatorPE alojados en el nodo 1 están continuamente enviando eventos a EvaluatorPE, por lo que una caída del nodo 2 debería detectarse fácilmente.

Paramos el nodo 2 por consola, y observamos la salida del nodo 1, que no tarda en mostrar lo siguiente:

```
WARN  org.apache.s4.comm.tcp.TCPEmitter: Failed to send message to
node {partition=1,port=12001,machineName=ubuntu-server,taskId=Task-
1} (according to current cluster information)
```

Zookeeper detecta la caída del nodo 2, e instantáneamente redibuja la topología del clúster, como quedará reflejado en la salida del nodo 1 (en negrita):

```
INFO  o.a.s4.comm.topology.ClusterFromZK: Changing cluster topology
to {
nbNodes=1,name=cluster1,mode=unicast,type=,nodes=[{partition=0,port
=12000,machineName=ubuntu-server,taskId=Task-0}]} from {
nbNodes=2,name=cluster1,mode=unicast,type=,nodes=[{partition=0,port
=12000,machineName=ubuntu-server,taskId=Task-0},
{partition=1,port=12001,machineName=ubuntu-server,taskId=Task-1}]}
```

Con esto queda constancia de la capacidad de S4 para detectar los nodos caídos y reponerse rápidamente ante la situación.

6.3.2. Activación de nodo de respaldo

S4 podrá reemplazar el nodo caído si tenemos algún otro en espera; es decir, si hemos arrancado más nodos que particiones tiene el clúster. De este modo el funcionamiento y rendimiento global no se vería afectado.

Para probar dicho comportamiento, arrancamos un nodo extra en el primer clúster (superando el número de particiones definidas para el mismo). El resultado de esta ejecución difiere del obtenido con los nodos anteriores, ya que se nos informa de que el nodo queda en espera:

```
INFO  o.a.s.comm.topology.AssignmentFromZK: Could not acquire task.
Going into standby mode
```

Seguiremos los mismos pasos del procedimiento para ejecutar nuestra aplicación con una partición y un nodo activo en cada clúster. Una vez funcionando, pararemos el nodo activo en el clúster principal, y después de un corto espacio de tiempo, observaremos lo siguiente en el nodo 2:

```
INFO  o.a.s4.comm.topology.ClusterFromZK: Changing cluster topology
to { nbNodes=0,name=unknown,mode=unicast,type=,nodes=[]} from {
nbNodes=1,name=cluster1,mode=unicast,type=,nodes=[{partition=0,port
=12000,machineName=ubuntu-server,taskId=Task-0}]}
INFO  o.a.s.comm.topology.AssignmentFromZK: Successfully acquired
task:Task-0 by ubuntu-server
02:54:10.247 INFO  o.a.s4.comm.topology.ClusterFromZK: Changing
cluster topology to {
nbNodes=1,name=cluster1,mode=unicast,type=,nodes=[{partition=0,port
=12000,machineName=ubuntu-server,taskId=Task-0}]} from {
nbNodes=1,name=cluster1,mode=unicast,type=,nodes=[{partition=0,port
=12000,machineName=ubuntu-server,taskId=Task-0}]}
INFO  o.a.s4.comm.topology.ClustersFromZK: Detected new stream
[Document]
INFO  org.apache.s4.core.S4Bootstrap: Loaded application from file
/tmp/tmp690498508266565621 5s4r
INFO  org.apache.s4.core.App: Init prototype
[es.uc3m.it.jduran.disambiguator.EvaluatorPE].
INFO  org.apache.s4.core.App: Init prototype
[es.uc3m.it.jduran.disambiguator.DisambiguatorPE].
INFO  org.apache.s4.core.App: Init prototype
[es.uc3m.it.jduran.disambiguator.ProbabilityCalculatorPE].
INFO  org.apache.s4.core.App: Init prototype
[es.uc3m.it.jduran.disambiguator.MetadataExtractorPE].
INFO  DisambiguatorPE:
[documentId=/home/webtlab/NYT/corpus/1987/01/03/0000460.xml]
Recibida probabilidad del candidato 1/2: new_york_state.loc => -
7.917171854735328
INFO  DisambiguatorPE:
[documentId=/home/webtlab/NYT/corpus/1987/01/03/0000460.xml]
Recibida probabilidad del candidato 2/2: new_jersey.loc => -
7.624618986159398
INFO  DisambiguatorPE:
[documentId=/home/webtlab/NYT/corpus/1987/01/03/0000460.xml]
Recibidas todas las probabilidades. Candidato elegido:
new_jersey.loc. Probabilidad:-7.624618986159398
INFO  EvaluatorPE:
[documentId=/home/webtlab/NYT/corpus/1987/01/03/0000460.xml]
Entidad desambiguada: new_jersey.loc -  Entidad correcta:
new_york_state.loc - FALLO!
```

```
INFO  EvaluatorPE: Estimaciones del desambiguador: 1
INFO  EvaluatorPE: Aciertos del desambiguador: 0
INFO  EvaluatorPE: Elecciones al azar del desambiguador: 0
INFO  EvaluatorPE: Efectividad del desambiguador: 0.00%

(...)
```

Como podemos observar, Zookeeper hace que el nodo que se encontraba en espera coja el relevo del nodo caído, y empiece a ejecutar sus mismas tareas instantáneamente. También vemos que se pierden las estadísticas anteriores de EvaluatorPE; esto se puede evitar, como ya vimos, gracias al uso de *checkpoints*.

6.3.3. Recuperación mediante checkpoints

Para poder recuperar el estado de los nodos caídos mediante *checkpoints*, tendremos que cumplir cuatro requisitos:

- Activar los *checkpoints* para los PE de nuestra aplicación; cosa que se hará en el código.
- Tener en cada PE donde se quieran activar:
  - Un constructor sin argumentos vacío
  - Un conjunto identificado de propiedades que definen el estado del PE, de tipo serializable, no definidas con la palabra clave *transient*.
- Tener un directorio compartido accesible desde todas las máquinas donde se guardarán los *checkpoints*.
- Arrancar los nodos con un parámetro adicional indicado a continuación.

Ejecutamos nuestra aplicación igual que en el apartado 5.2.3, arrancando más nodos de los que acepta el clúster principal activos, y modificando el siguiente paso:

7. Desplegar la aplicación disambiguator en el clúster principal con el mecanismo de *checkpoints* activado, especificando el módulo usado mediante el parámetro "-emc". Además con el parámetro "-p" indicaremos la ubicación del directorio compartido donde se guardarán los *checkpoints*. Existen más opciones; como se muestra en el apartado 4.3.4.2.
   Si lo hacemos en una máquina donde está corriendo Zookeeper, ejecutaremos lo siguiente:

```
./s4 deploy -s4r=$HOME/sharedspace/disambiguator.s4r -c=cluster1 -
appName=disambiguator
-emc=org.apache.s4.core.ft.FileSystemBackendCheckpointingModule -
p=s4.checkpointing.filesystem.storageRootPath=$HOME/sharedspace/che
ckpoints
```

Si lo hacemos en cualquier otra máquina:

```
./s4 deploy -s4r=$HOME/sharedspace/disambiguator.s4r -c=cluster1 -
appName=disambiguator -zk=<host_zookeeper>:2181 -emc=
org.apache.s4.core.ft.FileSystemBackendCheckpointingModule -p
s4.checkpointing.filesystem.storageRootPath=$HOME/sharedspace/check
points
```

Cuando hayamos finalizado todos los pasos, podemos observar que efectivamente se están guardando los *checkpoints* en el directorio que hemos señalado en el parámetro s4.checkpointing.filesystem.storageRootPath. Además con el nivel de logado a DEBUG veremos trazas como las siguientes en el terminal:

```
DEBUG ProcessingElement: Started checkpointing timer for PE
prototype
[es.uc3m.it.jduran.disambiguator.ProbabilityCalculatorPE], ID []
with interval [30] [SECONDS].

(...)


DEBUG DefaultFileSystemStateStorage: Checkpointing
[[PROTO_ID];[KEY] -->
[es.uc3m.it.jduran.disambiguator.EvaluatorPE];[result]] into file:
[/root/sharedspace/checkpoints/es.uc3m.it.jduran.disambiguator.Eval
uatorPE/W2VzLnVjM20uaXQuamR1cmFuLmRpc2FtYmlndWF0b3IuRXZhbHVhdG9yUEV
dO1tyZXN1bHRd]
```

La forma más eficiente para probar que usando *checkpoints* en este ejemplo no se pierde el estado del nodo caído es nuevamente parar el que contiene al elemento de procesado final, EvaluatorPE. En un breve instante a continuación arrancará de forma automática el nodo que teníamos en espera. Observaremos en la salida del nodo activado que se realiza recuperación de los *checkpoints* cuando se instancia un PE que tuviera estado previo:

```
DEBUG o.a.s.c.f.DefaultFileSystemStateStorage: Fetching
/root/sharedspace/checkpoints/es.uc3m.it.jduran.disambiguator.Metad
ataExtractorPE/W2VzLnVjM20uaXQuamR1cmFuLmRpc2FtYmlndWF0b3IuTWV0YWRh
dGFFeHRyYWN0b3JQRV07W3NpbmdsZXRvbl0for : [PROTO_ID];[KEY] -->
[es.uc3m.it.jduran.disambiguator.MetadataExtractorPE];[singleton]

DEBUG o.a.s.c.f.DefaultFileSystemStateStorage: Fetching
/root/sharedspace/checkpoints/es.uc3m.it.jduran.disambiguator.Proba
bilityCalculatorPE/W2VzLnVjM20uaXQuamR1cmFuLmRpc2FtYmlndWF0b3IuUHJv
YmFiaWxpdHlDYWxjdWxhdG9yUEVdO1tuZXdfeW9ya19zdGF0ZS5sb2Ndfor :
[PROTO_ID];[KEY] -->
[es.uc3m.it.jduran.disambiguator.ProbabilityCalculatorPE];[new_york
_state.loc]
```

```
DEBUG o.a.s.c.f.DefaultFileSystemStateStorage: Fetching
/root/sharedspace/checkpoints/es.uc3m.it.jduran.disambiguator.Evalu
atorPE/W2VzLnVjM20uaXQuamR1cmFuLmRpc2FtYmlndWF0b3IuRXZhbHVhdG9yUEVd
O1tyZXN1bHRdfor : [PROTO_ID];[KEY] -->
[es.uc3m.it.jduran.disambiguator.EvaluatorPE];[result]
```

Y si inspeccionamos el resumen lanzado por EvaluatorPE después de la primera desambiguación, observaremos que las estadísticas obtenidas antes del error no se han perdido, como ocurría en el caso de no usar *checkpoints*:

```
INFO  EvaluatorPE:
[documentId=/home/webtlab/NYT/corpus/1987/01/04/0000923.xml]
Entidad desambiguada: new_jersey.loc -  Entidad correcta:
new_jersey.loc - ACIERTO!
INFO  EvaluatorPE: Estimaciones del desambiguador: 31
INFO  EvaluatorPE: Aciertos del desambiguador: 21
INFO  EvaluatorPE: Efectividad del desambiguador: 67.74%
```

Queda comprobada por tanto la utilidad y eficiencia de este mecanismo.

Capítulo 7. Conclusiones

A lo largo de este proyecto se ha profundizado en distintos temas, lo que nos permite sacar conclusiones sobre varios aspectos. Aunque el objetivo final era claro y consistía en realizar una aplicación que implementara desambiguación de entidades, sobre un flujo de documentos de entrada generado en tiempo real, para llegar a ese punto hemos tenido que pasar por varias etapas en las que ha sido necesario ahondar en los diversos problemas que se iban presentando.

Como primera cuestión a tratar nos enfrentamos el estudio de las herramientas disponibles en un campo relativamente novedoso y donde todavía queda mucho por evolucionar; el procesamiento de grandes flujos de datos en tiempo real nos planteaba una buena lista de exigencias para la aplicación, haciendo necesario el uso de alguna herramienta contrastada en el manejo de este tipo de información. Por suerte, encontramos un par de plataformas de código abierto lo suficientemente maduras, que proporcionan una API completa para la creación de aplicaciones de usuario orientadas a consumir y procesar grandes *streams*. Podemos sacar como conclusión de esta fase que, a pesar de que el hueco que existía en este campo ya parece cubierto, todavía existe lugar para más plataformas de este tipo, y las ya existentes aún pueden mejorar sus prestaciones. Otra opción es que alguna de estas (Storm, S4) saque una gran ventaja y se convierta en una especie de Hadoop para datos consumidos en tiempo real, pero de momento no ha ocurrido.

Tras una breve comparación de las características y ventajas de cada uno de los *frameworks* barajados, nos decantamos por S4 ya que parecía la más indicada para desarrollar nuestra aplicación. Uno de sus puntos fuertes es la facilidad para implementar la funcionalidad sin preocuparse demasiado de otro tipo de cosas. De

hecho, no es complicado poner a andar una aplicación sencilla; no sólo desde el punto de vista de la implementación sino también desde el punto de vista del despliegue en entornos distribuidos. Para aplicaciones más complejas, como puede ser nuestro desambiguador, es posible que no se necesite usar muchos más métodos de los aquí expuestos.

Quizás hemos echado de menos una documentación algo más extensa del funcionamiento detallado de S4. Y no ocultamos que nos hemos encontrado problemas en varios puntos del desarrollo, e incluso *bugs* del propio S4. Por suerte todas estas dificultades se han ido salvando gracias principalmente a la lista de correo para usuarios. El uso de S4 como herramienta para lidiar con grandes cantidades de datos ha sido bastante satisfactorio: hemos sobrepasado las 300 desambiguaciones por segundo sin problemas, con una robustez muy alta en los nodos; en todas las horas de ejecución llevadas a cabo sólo hemos detectado problemas puntuales y ajenos a S4.

El otro campo donde hemos tenido que profundizar a la hora de llevar a cabo nuestra aplicación ha sido el relativo a la desambiguación de entidades y los clasificadores de texto. Este punto era clave para nuestro objetivo, ya que los resultados finales de eficiencia dependían de la estrategia implementada para desambiguar menciones en documentos de entrada. Una vez analizado el problema y el contexto donde tendría que actuar la aplicación, se estudiaron las opciones disponibles en este campo. En esta vertiente de nuestro proyecto no tuvimos demasiadas dudas; después de descartar la gran mayoría debido a los requisitos de nuestra aplicación elegimos el algoritmo más usado en los problemas de desambiguación de entidades en tiempo real, y el que mejor relación eficiencia/simplicidad aporta (lo ideal para procesamiento en tiempo real); el algoritmo de Naïve Bayes. En este proyecto hemos llevado a cabo su implementación siguiendo varias aproximaciones; siendo la más eficiente la versión que aplica suavizado de Laplace con unos mínimos ajustes adicionales. Hemos observado que es muy importante ajustarse al algoritmo; por experiencia, un pequeño cambio en uno de los factores de la fórmula puede empeorar los resultados de forma severa.

Para evaluar nuestra implementación hemos usado la técnica de *pseudo-names*; típica para probar algoritmos de desambiguación. Como hemos podido ver a tenor de los resultados recogidos en el apartado anterior, en general podemos afirmar que el algoritmo de Naïve Bayes produce resultados más que aceptables. La efectividad lograda es bastante alta incluso en casos que a priori eran considerados complicados para el algoritmo, como son aquellos en los que ambiguamos dos entidades que comparten el mismo contexto; es decir, muchas de las entidades junto a las que suelen aparecer en entrenamiento coinciden. La conclusión obtenida en este punto, aparte de que el algoritmo funciona razonablemente bien, es que la aplicación podría

perfectamente ser usada para ahorrar trabajo a los editores, como era nuestro cometido. Los resultados son aceptables incluso con pocas muestras de entrenamiento, aunque obviamente mejoran a medida que procesemos más artículos, ya que nuestra máquina aprende con cada uno de ellos, y las entidades del contexto se empiezan a repetir.

Otro de los puntos evaluados de la aplicación ha sido su respuesta ante errores. Teniendo algún nodo en espera es muy difícil perder una cantidad de datos apreciable. Zookeeper nos da una base robusta para cubrir el tema de la recuperación, y los *checkpoints* periódicos nos garantizan que no se pierde el estado previo de los PE.

Analizando todas estas conclusiones podemos afirmar que los objetivos propuestos al inicio de este proyecto han sido cumplidos y además con resultados muy satisfactorios. Aparte de esto hemos conseguido analizar y evaluar una plataforma genérica muy interesante para otros usos como es S4; obteniendo una excelente impresión de la misma.

Líneas futuras de trabajo

Debido a los requisitos necesarios para probar nuestra aplicación en un entorno más exigente y más próximo a lo que podría ser una plataforma en producción, no se ha podido llevar a cabo un estudio más exhaustivo del rendimiento; sobre todo de las características que recaen sobre S4 o Zookeeper, como pueden ser la alta eficiencia, el bajo retardo, o la alta disponibilidad. Es por ello que se plantea un estudio más exhaustivo de S4 y pruebas de la plataforma en un entorno más real o exigente. Así mismo sería interesante realizar la implementación del desambiguador en la plataforma Storm, para poder realizar una comparación punto por punto de ambas.

La aplicación implementada podría extenderse añadiendo otras etapas de procesamiento. Aquí sólo se realiza el proceso de desambiguación, que es sólo una parte de la extracción de entidades en documentos de texto. Podría añadirse una etapa anterior que tratase de detectar menciones a entidades, para luego poder almacenarlas y elaborar una lista de las correspondencias menciones-entidades. Este sería el punto de enganche a la aplicación desarrollada en el presente proyecto.

Otra posibilidad para ampliar la aplicación sería añadir más metadatos al proceso aparte de las entidades. Por ejemplo podríamos usar las categorías del artículo, sus etiquetas o el sello temporal para mejorar el proceso de desambiguación por contexto. Igualmente se podría hacer una implementación similar en otro sentido, para categorizar los artículos a partir de las entidades.

La elección de Naïve Bayes responde a varios criterios: su relación eficiencia/coste, el uso extendido en el ámbito de la clasificación de texto, y sobre todo la baja complejidad que supone, ideal para nuestro escenario. Podría realizarse una implementación de la aplicación usando los otros dos algoritmos señalados en el estado del arte (Árboles de Hoeffding y SPDT), y hacer una comparación de los resultados obtenidos; tanto de la efectividad, como de otros parámetros relativos a retardo y recursos utilizados.

Anexo A.
Planificación y Presupuesto

En este apartado trataremos de reflejar el tiempo invertido en el desarrollo del proyecto, así como un presupuesto estimado para su realización.

Planificación

El presente proyecto ha sido llevado a cabo por una persona que compaginaba la realización del mismo con su trabajo habitual. Es por ello que la planificación con fechas estrictas no era posible, y por tanto lo reflejado en este apartado se corresponde con la ocupación aproximada recogida a posteriori.

El calendario tiene en cuenta una jornada laboral de 4h diarias, de lunes a viernes, aunque en la práctica se ha realizado en buena parte durante los fines de semana, con la salvedad de los meses de verano, donde el autor contaba con jornada reducida en su trabajo habitual.

El proyecto se desarrolló entre julio del 2012 y enero del 2014, con una duración total de 18 meses. Podemos observar en la planificación varios intervalos de tiempo donde no se llevó a cabo ningún avance con el proyecto, debido a la falta de tiempo.

Las tareas se dividen en 3 grandes grupos:

1. Análisis
2. Implementación de la aplicación S4 del desambiguador
3. Memoria final
4. Segunda versión

Este es el desglose de tareas:

Nombre de tarea	Duración	Comienzo	Fin
ANÁLISIS			
Arranque del proyecto	1 día	mar 03/07/12	mar 03/07/12
Análisis del problema y estado del arte	10 días	mié 04/07/12	mar 17/07/12
Análisis de S4 y documentación	10 días	mié 18/07/12	mar 31/07/12
Instalación del software y configuración	5 días	lun 03/09/12	vie 07/09/12
Ejecución de HolaMundo	3 días	lun 10/09/12	mié 12/09/12
Ejecución de Aplicación ejemplo de Twitter	3 días	lun 01/10/12	mié 03/10/12
Pruebas de las distintas opciones de S4	5 días	lun 07/10/13	vie 11/10/13
Pruebas de despliegue distribuido	3 días	lun 15/10/12	mié 17/10/12
Pruebas de mecanismo de failover de S4	2 días	lun 22/10/12	mar 23/10/12
Implementación de checkpoints	3 días	lun 29/10/12	mié 31/10/12
Estudio de wiki de S4	5 días	lun 19/11/12	vie 23/11/12
Documentación	10 días	lun 03/12/12	vie 14/12/12
Perseguir bug de S4 (despliegue desde ubicación web)	2 días	lun 17/12/12	mar 18/12/12
APLICACIÓN S4 DEL DESAMBIGUADOR			
Especificación de requisitos para aplicación del Desambiguador	3 días	mié 19/12/12	vie 21/12/12
Arranque de la segunda fase	1 día	lun 25/02/13	lun 25/02/13
Diseño de la aplicación	6 días	mar 26/02/13	mar 05/03/13
Estudio de Algoritmo de Naïve Bayes	3 días	lun 11/03/13	mié 13/03/13
Implementación inicial de la aplicación	10 días	lun 18/03/13	vie 29/03/13
Implementación de adaptador con datos de prueba	3 días	lun 01/04/13	mié 03/04/13
Instalación de software adicional	3 días	lun 15/04/13	mié 17/04/13
Pruebas de funcionalidad	3 días	jue 18/04/13	lun 22/04/13
Implementación de adaptador con datos reales	3 días	mar 23/04/13	vie 26/04/13
Pruebas con un par de entidades ambiguadas	2 días	lun 06/05/13	mié 08/05/13
Ajustes de la aplicación para mejorar rendimiento	3 días	mié 08/05/13	lun 13/05/13
Análisis del corpus y extracción de 14 ejemplos para probar aplicación	1 día	lun 20/05/13	mar 21/05/13
Implementación de script para reducir corpus de entrada	2 días	mar 21/05/13	jue 23/05/13
Creación de corpus reducidos	1 día	jue 23/05/13	vie 24/05/13
Ejecución de la aplicación para los 14 ejemplos con el fin de ajustar el algoritmo de Naïve Bayes	10 días	lun 27/05/13	lun 10/06/13
Diseño de casos de prueba	2 días	lun 17/06/13	mié 19/06/13
Implementación y ejecución de casos de prueba	3 días	mié 19/06/13	lun 24/06/13
Análisis de los resultados y Ajustes de la aplicación	3 días	lun 08/07/13	jue 11/07/13
Actualización de S4 a versión 0.6.0	2 días	mié 17/07/13	vie 19/07/13

Actualización de aplicaciones a versión 0.6.0	3 días	lun 22/07/13	jue 25/07/13
Ajuste de logs de S4	2 días	lun 29/07/13	mié 31/07/13
Ejecución de la aplicación para los 14 ejemplos con el algoritmo ajustado	15 días	mié 14/08/13	mié 04/09/13
Persecución de bug de S4 (logs)	2 días	mié 04/09/13	vie 06/09/13
MEMORIA FINAL			
Análisis de los resultados	2 días	vie 06/09/13	mar 10/09/13
Documentación de la aplicación	5 días	mié 21/08/13	mié 28/08/13
Memoria final	20 días	mar 10/09/13	mar 08/10/13
SEGUNDA VERSIÓN			
Reajuste de algoritmo y repetición de pruebas	5 días	vie 15/11/13	vie 22/11/13
Reorganización de la memoria y ampliación de información	15 días	jue 26/12/13	mie 15/01/14
Total	**195 días**	**mar 03/07/12**	**mie 15/01/14**

Presupuesto

Para la realización del presupuesto, hemos trasladado el proyecto al ámbito de una empresa. Se han contabilizado los gastos de personal y de equipo/material amortizado durante de la duración del mismo, resultando el siguiente presupuesto.

_____ º _____

Autor

Jaime Durán Rodríguez

Departamento

Ingeniería Telemática

Descripción del proyecto

Título	Estudio de S4 y Aplicación práctica para la desambiguación de entidades en grandes streams de datos
Duración	18 meses
Tasa de costes indirectos	21 %

Presupuesto total del proyecto

25567,30 Euros

Desglose presupuestario (costes directos)

PERSONAL				
Recurso	**Categoría**	**Coste/Hora (€)**	**Horas**	**Coste total (€)**
Jaime Durán Rodríguez	Ingeniero	25	780	19.500
Norberto Fernández García	Doctor	40	100	4.000
			Total	23.500

EQUIPOS					
Descripción	**Coste (€)**	**% Uso dedicado**	**Dedicación (meses)**	**Periodo de depreciación**	**Coste imputable (€)**
Ordenador Portátil de trabajo	450	100	18	48	169
Servidor del departamento	750	40	12	60	60
Servidor del departamento	750	40	12	60	60
				Total	289

OTROS COSTES	
Descripción	**Coste imputable (€)**
Material de oficina	60
Electricidad	170
Total	230

Resumen de costes

Concepto	Presupuesto (€)
Personal	23.500
Equipos	289
Otros costes directos	230
Costes indirectos (21%)	5044
Total	**29063**

Anexo B.
Detalles de S4

B.1 COMANDOS Y OPCIONES DISPONIBLES

S4 dispone de varias instrucciones que nos permiten manejar todas sus opciones por completo desde línea de comandos. La forma de ejecutar estas órdenes sigue la sintaxis:

```
./s4 <comando> [opciones]
```

S4 en su versión 0.6.0 proporciona los siguientes 8 comandos básicos con una funcionalidad muy específica:

- **zkServer**: arranca una instancia de servidor de Zookeeper
- **newCluster**: crea un nuevo clúster, definiendo su tamaño y el puerto a partir del cual se establecerán sockets.
- **node**: arranca un nuevo nodo en un clúster.
- **s4r**: empaqueta una aplicación en un archivo .s4r con la intención de poder desplegarla después.
- **deploy**: despliega una aplicación en los nodos de un clúster determinado, especificando opcionalmente parámetros de configuración.
- **newApp**: crea el esqueleto de un nuevo proyecto.
- **status**: muestra el estado de clústeres, nodos, aplicaciones y *streams*.
- **adapter**: arranca un adaptador para inyección de datos.

Para recibir ayuda sobre cada uno de ellos, ejecutaremos lo siguiente:

```
./s4 <comando> -help
```

A continuación se ofrece una relación de las opciones permitidas para cada comando, resaltando en negrita las obligatorias:

zkServer

-clean	limpia los datos de ejecuciones anteriores de Zookeeper. Es recomendable siempre su uso
-clusters	lista de definiciones de clústeres, separadas por comas, con el siguiente formato: **c=**<nombre>**:flp=**<primer puerto de escucha>**:nbTasks=**<número de tareas>
-dataDir	directorio temporal para datos. Por defecto: /tmp/tmp/zookeeper/data
-gradleOpts	opciones para los scripts de Gradle (las mismas que se pueden especificar en la variable de entorno GRADLE_OPTS)
-logDir	directorio de *logs*. Por defecto /tmp/tmp/zookeeper/log
-port	puerto que ocupa Zookeeper. Por defecto el 2181
-t, -testMode	arranca la instancia de Zookeeper con un par de clústeres preconfigurados para pruebas rápidas (c=testCluster1:flp=12000:nbTasks=1 y c=testCluster2:flp=13000:nbTasks=1)

newCluster

-c, -cluster	nombre para el nuevo clúster
-flp, -firstListeningPort	puerto inicial de escucha para los nodos en el clúster. Cada nuevo nodo escuchará en el primer puerto libre a partir del especificado
-gradleOpts	opciones para los scripts de Gradle (las mismas que se pueden especificar en la variable de entorno GRADLE_OPTS)

-nbTasks	número de tareas o particiones para el clúster. Equivale al máximo de nodos que pueden estar activos en cada momento. Por defecto: 1
-zk	ubicación de Zookeeper siguiendo la forma: <máquina>:<puerto>. Por defecto: localhost:2181

node

-baseConfig	ruta a fichero de configuración de S4 para sustituir la configuración por defecto
-c, -cluster	nombre del clúster donde se crea el nodo
-p, -namedStringParameters	lista de parámetros de configuración con sus correspondientes valores, separados por comas, que sustituirán a los presentes en ficheros de configuración. Su sintaxis será: -p=<nombre1>=<valor1>,<nombre2>=<valor2>,… Nota: sólo se deberían incluir aquí parámetros relativos a los nodos
-zk	ubicación de Zookeeper siguiendo la forma: <máquina>:<puerto>. Por defecto: localhost:2181

s4r

-a, -appClass	nombre completo de la clase que alberga la aplicación. Debe ser subclase de App o AdapterApp
-b, -buildFile	ruta al fichero de *build* para Gradle de la aplicación S4
-debug	muestra información para depuración del proceso
-gradleOpts	opciones para los scripts de Gradle (las mismas que se pueden especificar en la variable de entorno GRADLE_OPTS)

deploy

-a, -appClass	nombre completo de la clase que alberga la aplicación. Debe ser subclase de App o AdapterApp

-appName	nombre de la aplicación S4
-c, -cluster	nombre del clúster sobre el que se despliega la aplicación
-debug	muestra información para depuración del despliegue
-gradleOpts	opciones para los scripts de Gradle (las mismas que se pueden especificar en la variable de entorno GRADLE_OPTS)
-mc, -emc, -moduleClasses	nombres completos de clases de módulos particulares de la aplicación
-mu, -moduleURIs	URIs para obtener código de módulos particulares de la aplicación
-p, -namedStringParameters	lista de parámetros de configuración con sus correspondientes valores, separados por comas, que sustituirán a los presentes en ficheros de configuración. Su sintaxis será: -p=<nombre1>=<valor1>,<nombre2>=<valor2>,…
-s4r	URI de un fichero .s4r existente con la aplicación empaquetada. Puede apuntar a una ruta local o a una ubicación web.
-timeout	tiempo máximo en ms para intentar conectar a Zookeeper. Por defecto 10000 ms.
-zk	ubicación de Zookeeper siguiendo la forma: <máquina>:<puerto>. Por defecto: localhost:2181

newApp

-gradleOpts	opciones para los scripts de Gradle (las mismas que se pueden especificar en la variable de entorno GRADLE_OPTS)
-parentDir	directorio padre de la aplicación. Por defecto el directorio principal de S4.

status

-app	muestra sólo información de las aplicaciones especificadas
-c, -cluster	muestra sólo información de los clústeres especificados
-gradleOpts	opciones para los scripts de Gradle (las mismas que se pueden especificar en la variable de entorno GRADLE_OPTS)
-s, -stream	muestra sólo información de los *streams* especificados
-timeout	tiempo máximo en ms para intentar conectar a Zookeeper. Por defecto 10000 ms.
-zk	ubicación de Zookeeper siguiendo la forma: <máquina>:<puerto>. Por defecto: localhost:2181

adapter

-c, -cluster	nombre del clúster donde se crea el adaptador
-commConfig	ruta a fichero de configuración de la capa de comunicación para sustituir la configuración por defecto
-commModuleClass	clase con el módulo de configuración para la capa de comunicación. Por defecto org.apache.s4.comm.DefaultCommModule
-coreConfig	ruta a fichero de configuración de S4 que sustituirá a la configuración por defecto del *core* de S4
-coreModuleClass	clase con el módulo de configuración para s4-core. Por defecto: org.apache.s4.core.DefaultCoreModule
-emc, -extraModulesClasses	lista de módulos adicionales de configuración separados por comas (serán instanciados a través de su constructor sin parámetros)
-p, -namedStringParameters	lista de parámetros de configuración con sus correspondientes valores, separados por comas, que sustituirán a los presentes en ficheros de configuración. Su sintaxis será:

	-p=<nombre1>=<valor1>,<nombre2>=<valor2>,… Nota: sólo se deberían incluir aquí parámetros relativos a nodos
-zk	ubicación de Zookeeper siguiendo la forma: <máquina>:<puerto>. Por defecto: localhost:2181

B.2 CONFIGURACIÓN ADICIONAL

Configuración de nodos

Los nodos obtienen su configuración (siempre que no se especifique por línea de comandos) de los siguientes ficheros en el *classpath*:

- default.s4.base.properties (configuración aplicada al arrancar el nodo)
- default.s4.comm.properties (configuración de la capa de comunicación)
- default.s4.core.properties (configuración global)

Podemos sobrescribir estas fuentes de propiedades usando la opción –p, que también nos deja modificar propiedades individualmente.

Configuración de aplicación y plataforma

En S4 podemos especificar parámetros de la aplicación y de la propia plataforma en el momento del despliegue. Por ejemplo:

- Clase principal de la aplicación.
- Ubicación del código de la aplicación.
- Módulos a utilizar.
- Ubicación de estos módulos.
- Parámetros para configurar esos módulos.

Un nodo S4 se compone de los siguientes módulos:

- Módulo base que especifica el modo de conectarse al mánager del clúster y la forma de descargar aplicaciones.
- Módulo de comunicaciones que especifica protocolos de comunicación, oyentes de eventos, y remitentes.
- Módulo *core* que especifica el mecanismo de despliegue de aplicaciones, el de serialización, etc.
- La aplicación desplegada en sí.

Sobrescribir módulos

Existe la posibilidad de usar otros módulos en lugar de los que proporciona S4 por defecto; podemos sobrescribir los existentes o crear los nuestros propios.

Si queremos activar o sustituir un módulo, lo especificaremos al hacer el despliegue con el comando *deploy*, usando la opción –emc (-moduleClasses). Podemos ver un ejemplo en el apartado 4.3.4.2, donde se ve cómo activar los *checkpoints* en nuestra aplicación.

Para escribir nuestros propios módulos, tendremos que empaquetarlos en un archivo JAR, y especificar en el despliegue la ubicación del mismo mediante la opción –mu (-modulesURIs). Por ejemplo:

```
./s4 deploy -c=cluster1
-emc=my.project.myCheckpointingModule
-mu=path/a/myCheckpointingModule.jar
```

Sobrescribir parámetros

Existe una manera muy simple de pasar parámetros a una aplicación:

a) Si los parámetros son particulares de nuestra aplicación, hay que inyectarlos en la clase principal de la misma mediante @Inject y @Named. Por ejemplo:

```
@Inject
@Named("redis")
public String redisConnectionString;
```

b) Especificarlos usando la opción –p en el arranque del nodo con el comando *node* (para parámetros de la plataforma) o en el despliegue con el comando *deploy* (para parámetros de la aplicación)

   Por ejemplo, para pasar el valor anterior a la aplicación:

```
./s4 deploy -s4r=path/a/fichero.s4r -appName=myApp
-p=redis=news.gast.it.uc3m.es:6379
```

   Otro ejemplo podemos verlo de nuevo en el apartado 4.3.4.2.

Paso de parámetros por fichero

Por comodidad podemos escribir un fichero de configuración y pasarle la ruta del mismo a un comando de S4 con la opción **@** en lugar de escribir todos los parámetros.

Ejemplo de ejecución:

```
./s4 deploy @/path/a/fichero/config
```

Contenido del fichero de ejemplo:

```
-s4r=path/a/app.s4r
  -c=cluster1
  -appName=myApp
  -emc=org.apache.s4.core.ft.FileSystemBackendCheckpointingModule
  -p=param1=value1,param2=value2
```

Configuración de checkpoints

El *framework* de *checkpoints* tiene los siguientes parámetros que podemos sobrescribir:

Serialización

- s4.checkpointing.serializationMaxThreads (default = 1)
- s4.checkpointing.serializationThreadKeepAliveSeconds (default = 120)
- s4.checkpointing.serializationMaxOutstandingRequests (default = 1000)

Almacenamiento

- s4.checkpointing.storageMaxThreads (default = 1)
- s4.checkpointing.storageThreadKeepAliveSeconds (default = 120)
- s4.checkpointing.storageMaxOutstandingRequests (default = 1000)

Consulta de PE

- s4.checkpointing.fetchingMaxThreads (default = 1)
- s4.checkpointing.fetchingThreadKeepAliveSeconds (default = 120)
- s4.checkpointing.fetchingMaxWaitMs (default = 1000) (tiempo para *timeout*)

Si el *backend* no responde, se pueden tomar medidas:

- s4.checkpointing.fetchingMaxConsecutiveFailuresBeforeDisabling (default = 10)
- s4.checkpointing.fetchingDisabledDurationMs (default = 600000)

B.3 EJECUCIÓN DE LA APLICACIÓN HOLA MUNDO

1. Iniciar una instancia de Zookeeper mediante el comando *zkServer*:

```
./s4 zkServer -clean

[main] INFO  org.apache.s4.tools.ZKServer - Starting zookeeper
server on port [2181]
[main] INFO  org.apache.s4.tools.ZKServer - cleaning existing data
in [/tmp/tmp/zookeeper/data] and [/tmp/tmp/zookeeper/log]
[main] INFO  org.apache.s4.tools.ZKServer - Zookeeper server
started on port [2181]
```

2. Crear el clúster principal (donde residirá la aplicación), por ejemplo con 2 particiones (también llamadas tareas) escuchando en los puertos a partir del 12000. Para ellos usamos el comando *newCluster*:

```
./s4 newCluster -c=cluster1 -nbTasks=2 -flp=12000

[main] INFO  org.apache.s4.tools.DefineCluster - preparing new
cluster [cluster1] with [2] node(s)
[main] INFO  org.apache.s4.tools.DefineCluster - New cluster
configuration uploaded into zookeeper
```

Podemos combinar los pasos 2 y 3 en uno solo con la siguiente instrucción:

```
./s4 zkServer -clusters=c=cluster1:flp=12000:nbTasks=2 -clean
```

3. Arrancar 2 nodos S4 usando el comando *node*, con la configuración por defecto, en terminales distintos para poder ver su salida en paralelo. Este procedimiento será distinto cuando tengamos varias máquinas y ejecutemos la plataforma en modo distribuido.

```
./s4 node -c=cluster1

[main] INFO  org.apache.s4.core.Main - Initializing S4 node with:
- comm module class [org.apache.s4.comm.DefaultCommModule]
- comm configuration file [default.s4.comm.properties from
classpath]
- core module class [org.apache.s4.core.DefaultCoreModule]
- core configuration file[default.s4.core.properties from
classpath]
- extra modules: []
- inline parameters: []
[main] INFO  org.apache.s4.core.Main - Starting S4 node. This node
will automatically download applications published for the cluster
it belongs to
[main] INFO  o.a.s.comm.topology.AssignmentFromZK - New
session:88692999716012036; state is : SyncConnected
```

```
[main] INFO  o.a.s.comm.topology.AssignmentFromZK - Successfully
acquired task:Task-0 by infoflex
```

5. Movernos al directorio de nuestra aplicación y empaquetar la misma mediante el comando *s4r*:

```
cd workspace/helloWorld
./s4 s4r -a=hello.HelloApp -b=`pwd`/build.gradle helloWorld

(...)
BUILD SUCCESSFUL

Total time: 4.559 secs
```

6. Desplegar la aplicación mediante el comando *deploy*:

```
./s4 deploy -s4r=`pwd`/build/libs/helloWorld.s4r -c=cluster1 -
appName=helloWorld

INFO  org.apache.s4.tools.Deploy: Using specified S4R
[file:/home/jduran/s4-
0.6.0/workspace/helloWorld/build/libs/helloWorld.s4r]
```

Veremos cómo esto provoca que los 2 nodos del clúster S4 detecten la nueva aplicación, la carguen y la arranquen:

```
INFO  o.a.s.comm.topology.AssignmentFromZK: New
session:90392508883664899; state is : SyncConnected
INFO  o.a.s.comm.topology.AssignmentFromZK: Successfully acquired
task:Task-1 by news
INFO  org.apache.s4.core.S4Bootstrap: Initializing S4 app with :
[]]
INFO  org.apache.s4.core.S4Bootstrap: Loading application
[helloWorld] from file [/tmp/tmp6677335471442746590s4r]
WARN  org.apache.s4.core.DefaultCoreModule: s4.tmp.dir not
specified, using temporary directory [/tmp/1379280337516-0] for
unpacking S4R. You may want to specify a parent non-temporary
directory.
INFO  o.a.s4.base.util.S4RLoaderFactory: Unzipping S4R archive in
[/tmp/1379280337516-0/tmp6677335471442746590s4r-1379280337570]
INFO  org.apache.s4.core.S4Bootstrap: App class name is:
hello.HelloApp
INFO  o.a.s4.comm.topology.ClusterFromZK: Changing cluster topology
to {
nbNodes=2,name=cluster1,mode=unicast,type=,nodes=[{partition=0,port
=12000,machineName=news,taskId=Task-0},
{partition=1,port=12001,machineName=news,taskId=Task-1}]} from null
INFO  o.a.s4.comm.topology.ClusterFromZK: Adding topology change
listener:org.apache.s4.comm.tcp.TCPEmitter@33f98d58
INFO  o.a.s.c.s.ThrottlingSenderExecutorServiceFactory: Creating a
throttling executor with a pool size of 1 and max rate of 200000
events / s
```

```
INFO  org.apache.s4.core.util.S4Metrics: Metrics reporting not
configured
INFO  o.a.s4.comm.topology.ClusterFromZK: Adding topology change
listener:org.apache.s4.core.SenderImpl@512d8ecd
INFO  o.a.s4.comm.topology.ClustersFromZK: New
session:90392508883664899
INFO  o.a.s4.comm.topology.ClustersFromZK: New
session:90392508883664899
INFO  o.a.s4.comm.topology.ClusterFromZK: Changing cluster topology
to {
nbNodes=2,name=cluster1,mode=unicast,type=,nodes=[{partition=0,port
=12000,machineName=news,taskId=Task-0},
{partition=1,port=12001,machineName=news,taskId=Task-1}]} from null
INFO  org.apache.s4.core.S4Bootstrap: Loaded application from file
/tmp/tmp6677335471442746590s4r
DEBUG o.a.s4.comm.topology.ClustersFromZK: Adding input stream
[names] in cluster [cluster1]
INFO  o.a.s4.comm.topology.ClustersFromZK: Detected new stream
[names]
INFO  org.apache.s4.core.App: Init prototype [hello.HelloPE].
```

Para un entorno distribuido (varias máquinas) tendremos que mover el fichero .s4r a la ubicación deseada antes de hacer el despliegue, e indicar como opción del comando *deploy* la nueva ubicación; que será un directorio compartido o un servidor web accesible desde todos los nodos del clúster.

Lo único que falta llegados a este punto es el tráfico de entrada. Podemos inyectarlo a través de un adaptador. En la aplicación de ejemplo, el adaptador es una clase muy básica, que extiende a AdapterApp, escucha en un socket de entrada abierto en el puerto 15000, y convierte cada línea de caracteres recibida en un evento genérico de S4, donde el texto se guarda en un campo "name".

7. Primero necesitamos crear un nuevo clúster para ejecutar esta aplicación. Sólo necesitamos que tenga una partición, para albergar un nodo:

```
./s4 newCluster -c=cluster2 -nbTasks=1 -flp=13000

[main] INFO  org.apache.s4.tools.DefineCluster - preparing new
cluster [cluster2] with [1] node(s)
[main] INFO  org.apache.s4.tools.DefineCluster - New cluster
configuration uploaded into zookeeper
```

8. A continuación, desplegamos el adaptador mediante el comando *deploy*. Debemos especificar la clase del mismo, el nombre del *stream* de salida (que será el mismo nombre del *stream* de entrada esperado por la aplicación que queremos probar), el clúster donde se despliega la aplicación y el nombre de la misma.

```
./s4 deploy -appClass=hello.HelloInputAdapter -
p=s4.adapter.output.stream=names -c=cluster2 -appName=adapter

INFO  o.a.s.c.u.ParsingUtils$InlineConfigParameterConverter:
processing inline configuration parameter
s4.adapter.output.stream=names
INFO  org.apache.s4.tools.Deploy: No S4R path specified, nor build
file specified: this assumes the app is in the classpath
```

9. Sólo nos queda ejecutar el adaptador mediante el comando *adapter*, que es especial para esta tarea (no hay empaquetado ni copia de paquete). Ejecutaremos lo siguiente en el directorio raíz de nuestro proyecto:

```
./s4 adapter -c=cluster2

BUILD SUCCESSFUL

Total time: 2.824 secs
[main] INFO  o.a.s.comm.topology.AssignmentFromZK - New
session:90392508883664906; state is : SyncConnected
[main] INFO  o.a.s.comm.topology.AssignmentFromZK - Successfully
acquired task:Task-0 by news
[S4 platform loader] INFO  org.apache.s4.core.S4Bootstrap -
Initializing S4 app with : []]
[S4 platform loader] DEBUG org.apache.s4.core.S4Bootstrap - Adding
named parameters for injection : s4.adapter.output.stream=names,
[S4 platform loader] INFO  org.apache.s4.core.S4Bootstrap -
Starting S4 app with application class [hello.HelloInputAdapter]
[S4 platform loader] INFO  o.a.s4.comm.topology.ClusterFromZK -
Changing cluster topology to {
nbNodes=1,name=cluster2,mode=unicast,type=,nodes=[{partition=0,port
=13000,machineName=news,taskId=Task-0}]} from null
[S4 platform loader] INFO  o.a.s4.comm.topology.ClusterFromZK -
Adding topology change
listener:org.apache.s4.comm.tcp.TCPEmitter@8497904
[S4 platform loader] INFO
o.a.s.c.s.ThrottlingSenderExecutorServiceFactory - Creating a
throttling executor with a pool size of 1 and max rate of 200000
events / s
[S4 platform loader] INFO  org.apache.s4.core.util.S4Metrics -
Metrics reporting not configured
[S4 platform loader] INFO  o.a.s4.comm.topology.ClusterFromZK -
Adding topology change
listener:org.apache.s4.core.SenderImpl@2658dd2d
[S4 platform loader] INFO  o.a.s4.comm.topology.ClustersFromZK -
New session:90392508883664906
[S4 platform loader] INFO  o.a.s4.comm.topology.ClustersFromZK -
Detected new stream [names]
[S4 platform loader] INFO  o.a.s4.comm.topology.ClustersFromZK -
New session:90392508883664906
[S4 platform loader] INFO  o.a.s4.comm.topology.ClusterFromZK -
Changing cluster topology to {
nbNodes=2,name=cluster1,mode=unicast,type=,nodes=[{partition=0,port
=12000,machineName=news,taskId=Task-0},
{partition=1,port=12001,machineName=news,taskId=Task-1}]} from null
[S4 platform loader] INFO  o.a.s4.comm.topology.ClusterFromZK -
Changing cluster topology to {
```

```
nbNodes=1,name=cluster2,mode=unicast,type=,nodes=[{partition=0,port
=13000,machineName=news,taskId=Task-0}]} from null
[S4 platform loader] DEBUG o.a.s4.comm.topology.ClustersFromZK -
Adding output stream [names] in cluster [cluster2]
```

10. Para probar el funcionamiento de la aplicación, hay que mandar algo al puerto 15000, que es donde está escuchando nuestro adaptador:

```
echo "Jaime" | nc localhost 15000
```

En el terminal donde se ejecuta el adaptador veremos lo siguiente:

```
read: Jaime
```

Y en uno de los 2 nodos del primer clúster, aparecerá la salida:

```
Hello Jaime!
```

Si seguimos mandando mensajes, los nodos se alternarán en el saludo. Si se repite el texto de entrada, el saludo cambiará a "Hello again …".

Esto pone de manifiesto lo sencillo que es ejecutar la aplicación más básica.

B.4 EJECUCIÓN DE LA APLICACIÓN TWITTER TRENDING TOPICS

Podemos perfectamente ejecutar una aplicación S4 en una sola máquina. Sin embargo, una de las grandes ventajas de usar esta plataforma es la posibilidad de realizar una ejecución distribuida usando más máquinas, para contar así con más recursos (memoria RAM principalmente) y capacidad de procesamiento. Esto será extremadamente sencillo, debido a que S4 está concebida expresamente para ello.

A continuación se describe paso a paso el sencillo proceso para desplegar la aplicación de ejemplo descrita en el apartado 2.4.3 sobre un entorno distribuido (2 máquinas):

1. Crear un fichero twitter4j.properties en la máquina donde se va a ejecutar Zookeeper, dentro de nuestro directorio local, con este contenido (*debug* es opcional):

```
debug=true
oauth.consumerKey=<twitter app consumer key>
```

```
oauth.consumerSecret=<matching app consumer secret>
oauth.accessToken=<twitter account access token>
oauth.accessTokenSecret=<matching account access token secret>
```

Para obtener estas claves, tendremos que acceder a Twitter Developers Apps [62], registrar nuestra aplicación y generar las claves necesarias.

2. Arrancar una instancia de Zookeeper ejecutando lo siguiente en el directorio base de S4:

```
./s4 zkServer -clean &
```

Si queremos alta disponibilidad, y hemos instalado Zookeeper en todos nuestros servidores, ejecutaremos lo siguiente:

```
bin/zkServer.sh start
```

3. Definir 2 clústeres; uno para la aplicación en sí, y otro para el adaptador:

```
./s4 newCluster -c=cluster1 -nbTasks=3 -flp=12000;
./s4 newCluster -c=cluster2 -nbTasks=1 -flp=13000;
```

4. Arrancar nodos en el primer clúster. Si lo hacemos en una máquina donde está corriendo Zookeeper, ejecutaremos lo siguiente:

```
./s4 node -c=cluster1
```

Si lo hacemos en cualquier otra máquina, usaremos el parámetro “zk” para indicar el servidor y el puerto (2181 por defecto) donde se esté ejecutando el Zookeeper:

```
./s4 node -c=cluster1 -zk=<host_zookeeper>:2181
```

5. Arrancar nodo para el adaptador (ejemplo para misma máquina)

```
./s4 node -c=cluster2
```

6. Crear en la máquina que ejecuta ZooKeeper los ficheros .s4r a desplegar. Entre empaquetado y despliegue necesitamos mover los paquetes a una ubicación accesible desde todas las máquinas:

```
./s4 s4r -b=`pwd`/test-apps/twitter-counter/build.gradle -
appClass=org.apache.s4.example.twitter.TwitterCounterApp twitter-
counter
./s4 s4r -b=`pwd`/test-apps/twitter-adapter/build.gradle -
appClass=org.apache.s4.example.twitter.TwitterInputAdapter twitter-
adapter
```

Sólo nos queda mover los ficheros a la ubicación elegida (ya sea un directorio compartido o un servidor web). Si los dejamos en un directorio compartido basta con moverlos por línea de comandos:

```
mv test-apps/twitter-counter/build/libs/twitter-counter.s4r
$HOME/sharedspace/
mv test-apps/twitter-adapter/build/libs/twitter-adapter.s4r
$HOME/sharedspace/
```

7. Desplegar la aplicación twitter-counter en el clúster 1 (se hará automáticamente en todos sus nodos a pesar de estar en distintas máquinas). Si la ubicación del fichero .s4r es un directorio compartido:

```
./s4 deploy -appName=twitter-counter
-s4r=$HOME/sharedspace/twitter-counter.s4r -c=cluster1
```

Si la ubicación es un servidor web:

```
./s4 deploy -appName=twitter-counter
-s4r=http://<servidor>/<ruta>/twitter-counter.s4r -c=cluster1
```

8. Desplegar la aplicación twitter-adapter en el clúster 2.
   Si la ubicación del fichero .s4r es un directorio compartido:

```
./s4 deploy -appName=twitter-adapter
-s4r=$HOME/sharedspace/twitter-adapter.s4r -c=cluster2
-p=s4.adapter.output.stream=RawStatus
```

Si la ubicación es un servidor web:

```
./s4 deploy -appName=twitter-adapter
-s4r= http://<servidor>/<ruta>/twitter-adapter.s4r -c=cluster2
-p=s4.adapter.output.stream=RawStatus
```

En este momento se empiezan a recoger datos de Twitter y a adaptarlos para alimentar el *stream* de entrada con eventos que la aplicación empezará a procesar al momento.

9. Observamos nuestros *trending topics* gracias al fichero que se generará en uno de los nodos del clúster 1 (sabremos cuál porque en la salida del nodo se vuelca el contenido de dicho fichero cada 10 segundos):

```
tail -f TopNTopics.txt

topic [CallACutieOut] count [104]
topic [PeoplesChoice.] count [54]
topic [followmejennette] count [32]
topic [TuCaraMeSuena8] count [31]
topic [LQSA] count [28]
topic [CallOutACutie] count [25]
topic [gameinsight] count [23]
topic [TeamFollowBack] count [20]
topic [ThingsMoreHarmfulThanMarijuana] count [18]
topic [RT] count [16]
```

B.5 REDESPLIEGUE DE UNA APLICACIÓN

Actualmente no existe en S4 un comando para eliminar una aplicación actualmente desplegada en un clúster, por lo que para poder redesplegarla, seguiremos los siguientes pasos:

1. Matar todos los nodos en el clúster. Si tenemos un terminal para cada uno, basta con ejecutar Ctrl-C en todos ellos.
2. Matar el servidor Zookeeper y arrancarlo de nuevo con la opción *-clean*. Si no es posible de esta manera, eliminar el sub-árbol de Zookeeper correspondiente al clúster (usando el cliente de Zookeeper).
3. Crear de nuevo todos los clústeres, con sus nodos arrancados.
4. Desplegar la aplicación (no hace falta volver a empaquetarla a no ser que haya cambios en los ficheros fuentes)

Anexo C.
Software adicional

C.1 INSTALACIÓN DE JAVA

Como prerrequisito esencial para poder ejecutar S4 necesitamos tener un JDK instalado en nuestra máquina. Para ello descargaremos la versión más adecuada desde la página web de Oracle [63] (en el caso aquí reflejado, el binario de la versión 6 para 64 bits), y la pondremos en el directorio raíz de nuestra cuenta. Ejecutaremos los siguientes comandos:

```
chmod 755 jdk-6u45-linux-x64.bin
./jdk-6u45-linux-x64.bin
```

Para incluirlo persistentemente en el PATH y definir la variable JAVA_HOME, modificaremos el fichero .bash_profile en el directorio $HOME de nuestro usuario:

```
vi ~/.bash_profile
```

Y dentro del fichero incluiremos las siguientes líneas:

```
export JAVA_HOME=$HOME/jdk1.6.0_45
export PATH=$PATH:$JAVA_HOME/bin
```

Guardamos el fichero y ejecutamos lo siguiente para hacerlo efectivo sin tener que volver a hacer *login*:

```
source ~/.bash_profile
```

Ejecutamos la siguiente prueba para comprobar que el cambio se ha hecho efectivo:

```
java -version

java version "1.6.0_45"
Java(TM) SE Runtime Environment (build 1.6.0_45-b06)
Java HotSpot(TM) 64-Bit Server VM (build 20.45-b01, mixed mode)
```

Si el cambio no ha sido satisfactorio, revisaremos las rutas. En caso de haber instalado java en nuestra cuenta de usuario, podemos tener un conflicto con la versión global. Para subsanarlo, podemos ejecutar o meter en el fichero .bash_profile el siguiente comando:

```
alias java=$JAVA_HOME/bin/java
```

C.2 INSTALACIÓN DE GRADLE

Cuando descargamos la versión de S4 que contiene los fuentes en lugar de los binarios, necesitamos compilar y empaquetar la aplicación (lo que también se conoce como *build*). También lo necesitaremos para hacer lo propio con nuestras aplicaciones. Para ello usaremos Gradle, que es el sistema que usa S4.

Instalaremos Gradle en todos los servidores donde vaya a ejecutarse S4. Elegimos la versión 1.4, que es la recomendada para s4 0.6.0:

```
wget http://services.gradle.org/distributions/gradle-1.4-bin.zip
unzip gradle-1.4-bin.zip
```

Creamos el *wrapper* para S4. Para ello tendremos que movernos al directorio raíz de S4 y ejecutar *gradle wrapper*:

```
cd s4-0.6.0
<path>/gradle-1.4/bin/gradle wrapper
```

Esto generará los ficheros gradlew y gradlew.bat necesarios en el directorio de S4.

C.3 INSTALACIÓN DE REDIS

Para la aplicación del desambiguador, usamos Redis como base de datos con el fin de compartir información entre el adaptador de la entrada y el último PE del sistema.

Instalaremos Redis en el servidor donde se ejecute el adaptador (para alta disponibilidad necesitaríamos una instalación en clúster):

```
wget http://redis.googlecode.com/files/redis-2.6.13.tar.gz
tar xzf redis-2.6.13.tar.gz
cd redis-2.6.13
make
```

Para ejecutar el servidor:

```
<path>/redis-2.6.13/src/redis-server
```

Usamos la librería Jedis [57] para operar con la base de datos. La añadiremos en el fichero build.gradle de cada proyecto que la use:

```
project.ext["libraries"] = [
  …
  jedis:              'redis.clients:jedis:2.1.0'
]
```

```
dependencies {
   …
   compile (libraries.jedis)
}
```

C.4 INSTALACIÓN DE ZOOKEEPER

Para garantizar la alta disponibilidad en S4, es necesario que el servidor de Zookeeper no esté en una sola máquina, sino que esté presente a ser posible en todas, formando lo que se conoce como un *quorum*. Para ello no sirve con utilizar la versión de Zookeeper incluida con S4, sino que hay que hacer una instalación del propio *software*.

Es necesario ejecutar los siguientes pasos en cada uno de los servidores para poner el *quorum* en marcha [24]:

1. Descargar la aplicación y descomprimirla en el directorio deseado:

```
wget http://apache.rediris.es/zookeeper/zookeeper-3.4.5/zookeeper-3.4.5.tar.gz
tar -zxvf zookeeper-3.4.5.tar.gz
cd zookeeper-3.4.5
```

2. Editar configuración para incluir los servidores que forman parte del conjunto y dos puertos; uno para comunicación entre ellos y otro para elección del líder. Esta configuración es igual en todos los servidores:

```
cp conf/zoo_sample.cfg conf/zoo.cfg
vi conf/zoo.cfg

    # The number of milliseconds of each tick
    tickTime=2000

    # The number of ticks that the initial
    # synchronization phase can take
    initLimit=10

    # The number of ticks that can pass between
    # sending a request and getting an acknowledgement
    syncLimit=5

    # the directory where the snapshot is stored.
    # do not use /tmp for storage, /tmp here is just
    # example sakes.
    dataDir=/tmp/zookeeper

    # the port at which the clients will connect
    clientPort=2181

    # servers in zookeeper quorum
    server.1=<server1_ip>:2888:3888
    server.2=<server2_ip>:2888:3888
    server.3=<server3_ip>:2888:3888
```

3. Crear un fichero llamado “myid” en el directorio definido en configuración como “dataDir”, que contenga únicamente el número asignado al servidor en configuración. Por ejemplo, si estamos en el servidor identificado como “server.2”, el contenido del fichero /tmp/zookeeper/myid será:

```
 2
```

4. Para ejecutar Zookeeper, simplemente hay que ejecutar lo siguiente dentro del directorio donde se desplegó:

```
bin/zkServer.sh start
```

Los nodos S4 se intentarán conectar a su servidor Zookeeper local, y la sincronización entre ellos será transparente para la aplicación.

Anexo D. Referencias

D.1 BIBLIOGRAFÍA

[1] **Infographic: A day in the life of the Internet**. Sean Valant [en línea] Mayo 2013. Disponible en Internet: http://blog.hostgator.com/2013/05/02/a-day-in-the-life-of-the-internet/ [Consultado: Diciembre 2013]

[2] **Named Entity Definition**. Webknox [en línea]. Disponible en Internet: http://webknox.com/p/named-entity-definition [Consultado: Noviembre 2012]

[3] **Introducción al Aprendizaje Automático**. Jorge Díez, Juan José del Coz. Centro de Inteligencia Artificial, Universidad de Oviedo (Asturias, España) [en línea] Disponible en Internet: http://www.aic.uniovi.es/ssii/SSII-T7-IntroduccionAlAprendizaje.pdf [Consultado: Diciembre 2013]

[4] **Microblogging: Definición**. Elena Rodríguez, Sonia Menéndez, Pilar Martínez, Elena Martínez, Alba Pérez. Universidad Complutense (Madrid, España) [en línea] Enero 2012. Disponible en Internet: http://microblogging18.blogspot.com.es/2012/01/definicion.html [Consultado: Diciembre 2013]

[5] **Modelos de clasificación basados en Máquinas de Vectores Soporte**. Luis González. Departamento de Economía Aplicada I, Universidad de Sevilla (Andalucía, España) [publicación] 2003. Disponible en Internet: http://www.asepelt.org/ficheros/File/Anales/2003%20-%20Almeria/asepeltPDF/55.pdf [Consultado: Diciembre 2013]

[6] **Random Forests**. Leo Breiman. Statistics Department, Universidad de California (Berkeley, CA, EEUU). [publicación] 2001. Disponible en Internet: http://link.springer.com/content/pdf/10.1023%2FA%3A1010933404324.pdf [Consultado: Diciembre 2013]

[7] **Text Classification and Naïve Bayes**. Dan Jurafsky. Natural Language Processing Department, Universidad de Stanford (Palo Alto, CA, EEUU) [en línea] Mayo 2012. Disponible en Internet: http://www.stanford.edu/class/cs124/lec/naivebayes.pdf [Consultado: Abril 2013]

[8] **Mining High-Speed Data Streams**. Pedro Domingos, Geoff Hulten. Computer Science & Engineering, Universidad de Washington (Seattle, WA, EEUU) [publicación] 2000. Disponible en Internet: http://homes.cs.washington.edu/~pedrod/papers/kdd00.pdf [Consultado: Diciembre 2013]

[9] **A Streaming Parallel Decision Tree Algorithm**. Yael Ben-Haim, Elad Tom-Tov. IBM Haifa Research Lab (Israel) [publicación] Febrero 2010. Disponible en Internet: http://jmlr.org/papers/volume11/ben-haim10a/ben-haim10a.pdf [Consultado: Diciembre 2013]

[10] **Load Shedding**. Springer Reference [en línea]. Disponible en Internet: http://www.springerreference.com/docs/html/chapterdbid/63343.html [Consultado: Noviembre 2012]

[11] **Evaluación de la herramienta de código libre Apache Hadoop**. Mª Carmen Palacios Díaz-Zorita. Universidad Carlos III (Madrid, España) [proyecto fin de carrera] Noviembre 2011. Disponible en Internet: http://e-archivo.uc3m.es/bitstream/handle/10016/13533/MemoriaPFC_MCarmenPalacios.pdf [Consultado: Octubre 2013]

[12] **MapReduce: Simplified data processing on large clusters**. Jeffrey Dean, Sanjay Ghemawat. Google [en línea] Diciembre 2004. Disponible en internet: http://research.google.com/archive/mapreduce.html [Consultado: Julio 2013]

[13] **MapReduce Online**. Tyson Condy, Neil Conway, Peter Alvaro, Joseph Hellerstein, Khaled Elmeleegy, Rusell Sears. Universidad de California (Berkeley, CA, EEUU) [informe técnico] Octubre 2009. Disponible en Internet: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-136.pdf [Consultado: Octubre 2013]

[14] **SPC: A distributed, Scalable Platform for Data Mining.** Lisa Amini, Henrique Andrade, Ranjita Bhagwan, Frank Eskesen, Richard King, Philippe Selo, Yoonho Park, Chitra Venkatramani. IBM Research [en línea] 2006. Disponible en Internet: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.5471&rep=rep1&type=pdf [Consultado: Octubre 2013]

[15] **S4: Distributed Stream Computing Platform**. Leonardo Neumeyer, Bruce Robbins, Anish Nair, Anand Kesari. Yahoo! Labs (Santa Clara, CA, USA) [en línea] Noviembre 2010. Originalmente publicado por Yahoo!. Disponible en Internet: http://www.4lunas.org/pub/2010-s4.pdf [Consultado: Septiembre 2012]

[16] **Tech Talk: "S4: Distributed Stream Computing Platform"**. Leonardo Neumeyer, Anish Nair. Talks at Linkedin [vídeo en línea] Enero 2011. Disponible en Internet: http://vimeo.com/20489778 [Consultado: Septiembre 2012]

[17] **Actors: A model of concurrent computation in distributed systems**. Gul Abdulnabi Agha. MIT, Artificial Intelligence Laboratory (Cambridge, Massachusetts) [publicación] Junio 1985. Disponible en internet: http://dspace.mit.edu/handle/1721.1/6952 [Consultado: Agosto 2013]

[18] **Distributed stream processing showdown: S4 vs Storm**. Gianmarco De Francisci Morales [en línea] Enero 2013. Disponible en Internet: http://gdfm.me/2013/01/02/distributed-stream-processing-showdown-s4-vs-storm/ [Consultado: Octubre 2013]

[19] **S4 vs Storm**. Richard McCreadie. Glasgow University, CROSStreams Project (Escocia) [en línea] Mayo 2012. Disponible en Internet: http://demeter.inf.ed.ac.uk/cross/docs/s4vStorm.pdf [Consultado: Julio 2012]

[20] **¿Qué es la inyección de dependencias?** Jorge Rubira. Genbeta:dev [en línea] Abril 2011. Disponible en internet: http://www.genbetadev.com/paradigmas-de-programacion/que-es-la-inyeccion-de-dependencias [Consultado: Octubre 2013]

[21] **Design Pattern: Publish-Subscribe**. Riccardo Pucella. Northeastern University (Boston) [en línea] Abril 2010. Disponible en Internet: http://www.ccs.neu.edu/home/riccardo/courses/cs3500-sp10/lect21-publish-subscribe.pdf [Consultado: Octubre 2013]

[22] **Actor libraries and frameworks**. Wikipedia [en línea] Disponible en Internet: http://en.wikipedia.org/wiki/Actor_model#Actor_libraries_and_frameworks [Consultado: Octubre 2013]

[23] **Unsupervised Personal Name Disambiguation**. Gideon Mann, David Yarowsky. Johns Hopkins University, Department of Computer Science (Baltimore, MD, EEUU) [publicación] Mayo 2003. Disponible en Internet: http://dl.acm.org/citation.cfm?doid=1119176.1119181 [Consultado: Diciembre 2013]

[24] **Running replicated Zookeeper**. Documentación. The Apache Software Foundation [en línea] Disponible en Internet: http://zookeeper.apache.org/doc/r3.4.5/zookeeperStarted.html#sc_RunningReplicatedZooKeeper [Consultado: Noviembre 2013]

[25] **S4 v0.5.0 Documentation**. Wiki. The Apache Software Foundation [en línea] Agosto 2012. Disponible en Internet: https://cwiki.apache.org/confluence/display/S4/S4+Wiki [Consultado: Junio 2013]

[26] **S4 v0.6.0 Documentation**. Wiki. The Apache Software Foundation [en línea] Junio 2013. Disponible en Internet: http://incubator.apache.org/s4/doc/0.6.0/ [Consultado: Septiembre 2013]

D.2 OTRAS REFERENCIAS

[27] **Twitter**
https://twitter.com/ [Consultado: Septiembre 2013]

[28] **Eskup**
http://eskup.elpais.com/index.html [Consultado: Diciembre 2013]

[29] **Apache Hadoop**
http://hadoop.apache.org [Consultado: Septiembre 2013]

[30] **STREAM**
http://www.streamproject.eu [Consultado: Diciembre 2013]

[31] **Borealis**
http://cs.brown.edu/research/borealis/public/ [Consultado: Diciembre 2013]

[32] **Aurora**
http://cs.brown.edu/research/aurora/ [Consultado: Diciembre 2013]

[33] **Telegraph**
http://telegraph.cs.berkeley.edu/techover.html [Consultado: Diciembre 2013]

[34] **Apache Zookeeper**
http://zookeeper.apache.org/ [Consultado: Diciembre 2013]

[35] **Backtype**
http://www.backtype.com/ [Consultado: Diciembre 2013]

[36] **Storm**
http://storm-project.net/ [Consultado: Diciembre 2013]

[37] **Apache Samza**
http://samza.incubator.apache.org/ [Consultado: Diciembre 2013]

[38] **Linkedin**
https://www.linkedin.com/ [Consultado: Diciembre 2013]

[39] **Apache S4**
http://incubator.apache.org/s4/ [Consultado: Diciembre 2013]

[40] **Licencia Apache, v2.0**.
http://www.apache.org/licenses/LICENSE-2.0 [Consultado: Septiembre 2013]

[41] **JSON**
http://www.json.org/ [Consultado: Diciembre 2013]

[42] **Redis**
http://redis.io/ [Consultado: Diciembre 2013]
[43] **S4 v0.6.0 API javadoc**.
http://people.apache.org/~mmorel/apache-s4-0.6.0-incubating-doc/javadoc/ [Consultado: Septiembre 2013]
[44] **Repositorio Git de S4**.
https://github.com/apache/incubator-s4 [Consultado: Diciembre 2013]
[45] **Gradle**
http://www.gradle.org/ [Consultado: Diciembre 2013]
[46] **Maven**
http://maven.apache.org/ [Consultado: Diciembre 2013]
[47] **Eclipse**
http://www.eclipse.org/ [Consultado: Diciembre 2013]
[48] **IntellijIDEA**
http://www.jetbrains.com/idea/ [Consultado: Diciembre 2013]
[49] **Gson**
http://code.google.com/p/google-gson/ [Consultado: Diciembre 2013]
[50] **NFS**
http://recursostic.educacion.es/observatorio/web/es/software/software-general/733-nfs-sistema-de-archivos-de-red [Consultado: Diciembre 2013]
[51] **SSHFS**
http://fuse.sourceforge.net/sshfs.html [Consultado: Diciembre 2013]
[52] **RAID**
http://www.dlink.com/-/media/Files/B2B%20Briefs/ES/dlinkraid.pdf [Consultado: Diciembre 2013]
[53] **HDFS**
http://hadoop.apache.org/docs/stable1/hdfs_design.html [Consultado: Diciembre 2013]
[54] **Logback**
http://logback.qos.ch/ [Consultado: Diciembre 2013]
[55] **Manual de Logback**.
http://logback.qos.ch/manual/index.html [Consultado: Septiembre 2013]
[56] **Kryo**
http://code.google.com/p/kryo/ [Consultado: Diciembre 2013]
[57] **Jedis**
http://code.google.com/p/jedis/ [Consultado: Diciembre 2013]
[58] **Repositorio Central de Maven**.
http://search.maven.org [Consultado: Septiembre 2013]
[59] **Metrics**
http://metrics.codahale.com/ [Consultado: Diciembre 2013]

[60] **JMX**
http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html [Consultado: Diciembre 2013]

[61] **CSV**
http://www.creativyst.com/Doc/Articles/CSV/CSV01.htm [Consultado: Diciembre 2013]

[62] **Twitter Developer Apps**
https://dev.twitter.com/apps [Consultado: Septiembre 2013]

[63] **Oracle Java SE downloads**
http://www.oracle.com/technetwork/java/javase/downloads/index.html [Consultado: Diciembre 2013]

[64] **Parche S4-138 para S4 0.6.0**.
https://issues.apache.org/jira/browse/S4-138 [Consultado: 15 Septiembre 2013]